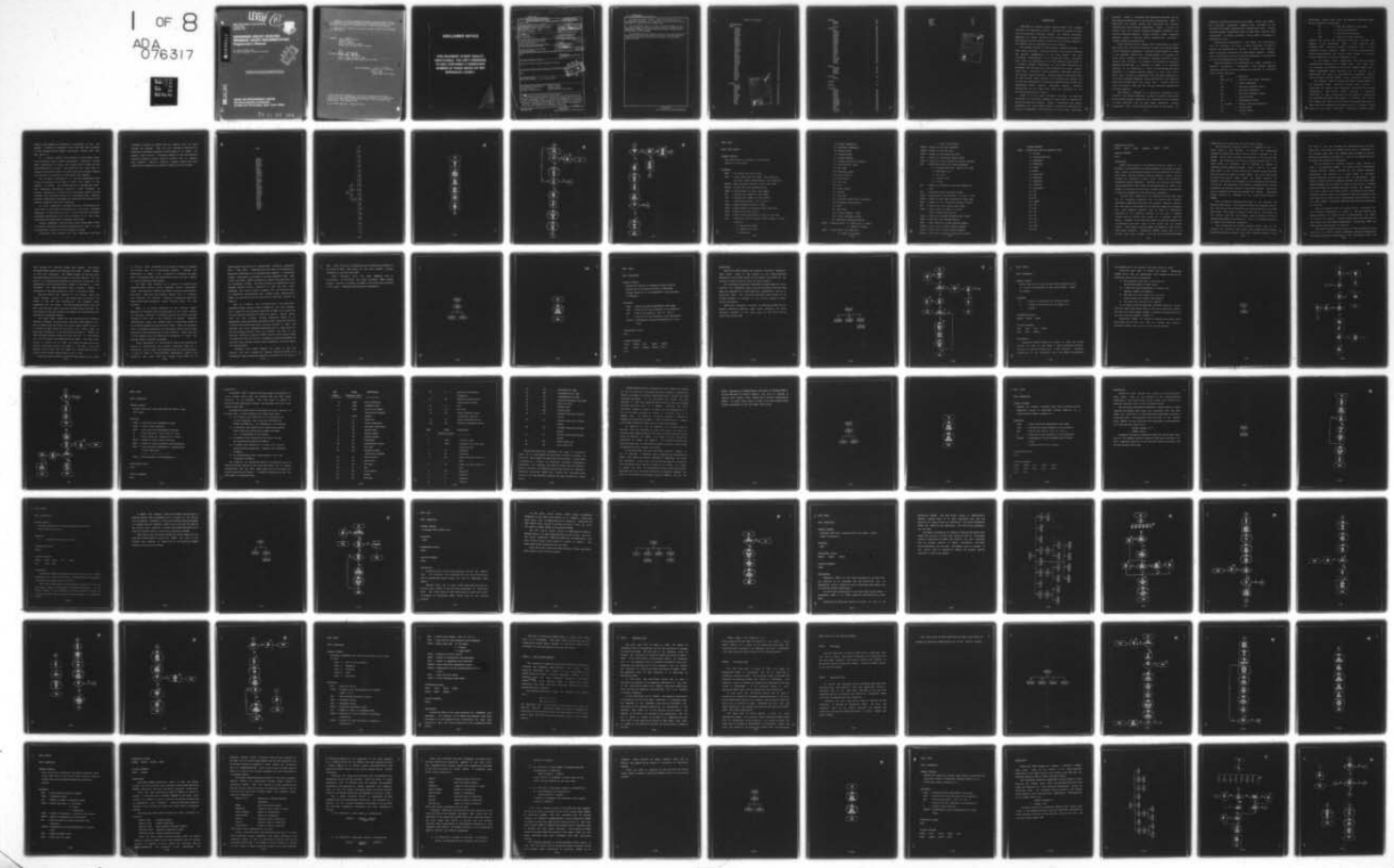
AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

1 OF 8
ADA 076317



LEVEL II

RADC-TR-79-245, Vol III (of three)
In-House Report
September 1979



NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION
Programmer's Manual

Mr. Jon B. Valente, RADC
Ms. Sharon Stratakos, Syracuse University

AD A076317

DDC
NOV 7 1979
E

Vol I
AD76384

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

79 11 07 074

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-245, Volume III (of three) has been reviewed and is approved for publication.

APPROVED: Samuel D. Zaccari

SAMUEL D. ZACCARI
Chief, Compatibility Branch
Reliability and Compatibility Division

APPROVED: David C. Luke

DAVID C. LUKE, Lt Colonel, USAF
Chief, Reliability and Compatibility Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (RBCT), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

(18) RADC (19) TR-79-245-VOL-3

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-79-245, Vol III (of three)

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER (9)

4. TITLE (and Subtitle): NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION Volume III. Programmer's Manual (6)

5. TYPE OF REPORT & PERIOD COVERED: In-House and Phase Report, May 78 — Jul 79

6. PERFORMING ORG. REPORT NUMBER: N/A

7. AUTHOR(s): Jon B. Valente, Sharon Stratakos (10)

8. CONTRACT OR GRANT NUMBER(s): In-House and Contract F30602-79-C-0011 (15) new

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Rome Air Development Center (RBCT), Griffiss AFB NY 13441; Syracuse University, Syracuse NY 13210

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F (17) 04; 23380314/23380317 (16)

11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (RBCT), Griffiss AFB NY 13441 (11)

12. REPORT DATE: September 1979

13. NUMBER OF PAGES: 711 (12) 731

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DDC RECEIVED NOV 7 1979 E

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Jon B. Valente (RBCT)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Electromagnetic Compatibility	Computer Program
Nonlinear Analysis	Volterra Analysis
Nonlinear Circuit Analysis	Computer-aided Circuit Analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Nonlinear Circuit Analysis Program (NCAP) is a circuit analysis code which uses the Volterra approach to solve for the transfer functions and node voltages of nonlinear circuits. To increase the transportability, the code was written in ANSI FORTRAN. The code was revised and documented in a joint effort using both in-house and contracted manpower. This documentation is a result of that effort. The documentation is made up of three volumes: the Engineering Manual, the User's Manual, and the Programmer's Manual.

(Cont'd)

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

339 600 JOB

The Engineering Manual, Volume I, contains the introduction to the documentation, a description of the theory upon which NCAP is based, and the procedure for obtaining the input parameter data.

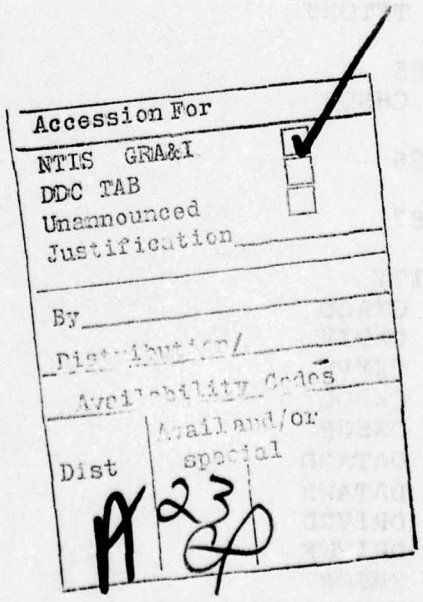
The User's Manual, Volume II, contains a detailed description of the NCAP input language. The description includes the input commands and data requirements. Examples are used throughout the manual to aid the user in understanding the input language.

The Programmer's Manual, Volume III, contains a subroutine by subroutine description of the code. Each subroutine has a trace map, functional flow diagram and narrative which will help the programmer to understnad the structure of NCAP.

Table of Contents

INTRODUCTION
MAIN 1
ALPHA 2
DATOUT 3
DECIF 4
DECIM 5
DRVOUT 6
ECIN 7
ENDIN 8
GENIN 9
GENOUT 10
INTEG 11
JFETIN 12
LCIN 13
LDSIN 14
MODGIN 15
MODIN 16
NCIN 17
NDSIN 18
NODEIN 19
PACKL 20
PACKR 21
PLOTIN 22
PRCSS 23
PSIN 24
PTIN 25
SCAN 26
SCIN 27
SDIN 28
SHIFT 29
TRNIN 30
VDIN 31
VTIN 32

PHASE1 33
CPMTJF 34
CPMTSD 35
CPMTVD 36
CPPT 37
CPT 38
CPVT 39
DERIDS 40
FIDS 41
IMLOC 42
MTPT 43
MTT 44
MTVT 45



PHASE2 46
MTLDS 47
MTNDS 48

ZIADD 49
PHASE3 50
BETA 51
CONTRL 52
CROSS 53
CURGEN 54
CURJF 55
CURNC 56
CURNDS 57
CURPT 58
CURSD 59
CURT 60
CURVD 61
CURVT 62
FNCTN 63
FRPRM 64
LOCTF 65
QFN 66
SSCODE 67
TRIANG 68
UPFRQ 69
UPSEQ 70

PHASE4 71
HHMMSS 72
PRNTIM 73
PRSET 74
TFTONV 75

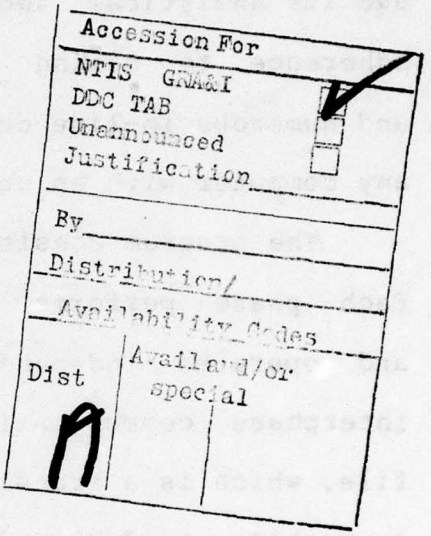
PHASE5 76
CHNGE 77

PHASE6 78

PHASE7 79

UTILITY
CXADD 80
CXDIV 81
CXMPY 82
CXPOL 83
CXSUB 84
DATARD 85
DATAWR 86
DRIVRD 87
DRIVWR 88
ERROR 89
FUN 90
LSHIFT 91
RSHIFT 92

SSCOD2 93
TFRD 94
TFWR 95
TIMOUT 96



INTRODUCTION

RADC NCAP is a Fortran program which computes the transfer functions of electronic circuits. Although the program is large and its analytical technique complex, the modular structure, adherence to naming conventions for subprograms and variables, and numerous in-line comments allow NCAP to be readily adapted to any computer with an appropriate Fortran compiler.

The program consists of eight phases, numbered 0 through 7. Each phase performs a distinct portion of the circuit analysis and operates independently of the other phases. The only interphase communication is by shared disk files: the driver file, which is a translation of the NCAP asterisk input cards to a machine readable description of the circuit analyses to be performed, and the data file, which contains all circuit element input data, calculated device parameters, admittance matrices, and transfer function vectors. Although several other disk files are used by NCAP, their function is to conserve core storage and to ease the transmission of internally generated data between the subprograms which comprise individual phases. Detailed descriptions of the NCAP disk files are contained in the narrative description of Phase 1.

Phase 0 is the input processor for NCAP. It reads and interprets the input deck, mapping the input cards to appropriate driver and data file records. Phase 1 calculates the device parameters for each circuit element, collects and tabulates the circuit's frequencies, and determines the size of the admittance

matrices. Phase 2 constructs the admittance matrices, one for each possible combination of the circuit's frequencies. Phase 3 constructs the current vectors and calculates the transfer functions for each frequency combination. Phase 4 prints the results from the circuit analysis performed in Phases 1-3 and controls frequency sweeping. Phase 5 controls linear component sweeping, Phase 6 controls device modification, and Phase 7 controls generator modification.

Since numerous circuit analyses may be specified by a single NCAP input deck, the path of execution through the program phases is not necessarily sequential. Execution always begins at Phase 0 and proceeds sequentially through Phases 1-4 to perform the first circuit analysis. From Phase 4, program execution either reverts back to Phase 1 to initiate a new analysis if frequency sweeping is specified, or proceeds to Phase 5 if frequency sweeping is not specified or after all such sweeps have been satisfied. In a similar fashion, Phases 5, 6, and 7 may either cycle back to Phase 1 or proceed on the next phase depending on the linear component sweeping, device modification, and generator modifications specified in the input deck. Program execution ends with Phase 7 after the last (if any) generator modification has been effected.

Each phase is composed of a principle subprogram which controls its general operation, a group of secondary subprograms which perform specific operations for individual circuit elements or NCAP functions, and in some cases, additional support subprograms which perform operations unique to that phase. The

program is organized sequentially by phases. Within each phase, the principle subprogram appears first, followed by the secondary and support subprograms in alphabetical order. A group of shared support subprograms, such as those which perform disk input/output or complex arithmetic, follow Phase 7 and appear in alphabetical order.

The principle subprograms of each phase are subroutines, with the exception of Phase Ø whose principle, in order to satisfy the requirements of Fortran, is NCAP's main program. These principle subroutines are named PHASEØ, PHASE1,...,PHASE7. With the exception of but two function subprograms, the remainder of the NCAP subprograms are subroutines.

Wherever possible, the subprograms are named according to specific conventions. Subprograms which perform specific functions related to circuit elements are prefixed or suffixied with a device identifier:

GEN	=	Generator
JFET (or JF)	=	Junction Field Effect Transistor
LC	=	Linear Components
LDS	=	Nonlinear Dependent Source
NC	=	Nonlinear Components
NDS	=	Nonlinear Dependent Source
PT	=	Vacuum Pentode
SD	=	Semiconductor Diode
T (or TRN)	=	Bipolar Junction Transistor
VD	=	Vacuum Diode
VT	=	Vacuum Triode

Furthermore, within each phase the secondary subprogram names contain functional identifiers:

IN	=	Read and interpret input cards
CP	=	Calculate parameters
MT	=	Create matrix elements
CUR	=	Calculate current elements

Together the device and functional identifier describe the purpose of the subprogram: GENIN = input generator card sequence, CPPT = calculate pentode parameters, MTT = create transistor matrix elements, CPMTVD = calculate parameters and create matrix elements for vacuum diode, CURNDS = calculate current elements for nonlinear dependent source.

At the support level, subprograms which perform complex arithmetic are prefixed by CX (CXADD, CXDIV, etc), while disk I/O routines are suffixed by RD and WR (DATARD, DRIVWR, etc.). Support subprograms whose functions are too specific to be categorized are named as descriptively as possible: LOCTF = locate transfer function, FRPRM = create frequency permutation.

The program code for subroutines PHASE1 through PHASE7 are all organized alike. Execution through these routines is controlled by reading and processing the driver file records sequentially. Each driver record contains a functional identifier or mode, which serves as the index of a computed GO TO, selecting the proper code segment to process that record. The coding for each driver function is arranged numerically by mode within the subroutine and begins with the statement number equal to the value of the mode. Additional statement numbers

within a code segment are assigned in increments of 100. For example, a section of transistor code would begin with statement 9 (the transistor driver mode), and proceed through 109, 209, 309, and so on.

In a similar fashion, the IN family of subroutines (input card processors) share a common organization. Execution through these subroutines is based on a computed GO TO using the card type identifier as an index. The coding for each card type is arranged numerically within the subroutines and statement numbers are allocated in increments of 100 within code segments.

The narrative descriptions of the NCAP subprograms which follow are arranged in the order in which they appear in the program· by phases, and within phases, by alphabetical order. Each subprogram description contains a brief statement of purpose, followed by a variable list, subroutines called, calling programs, and a detailed narrative of the program code. Wherever possible mathematical algorithms are summarized and tables of all possible computed results are presented.

To avoid repetition, variables which are used globally are listed only in the Phase 0 description or in the first principle subprogram in which they are used. In the secondary and support subprogram descriptions, only local variables (or in some cases less frequently used global variables) are listed.

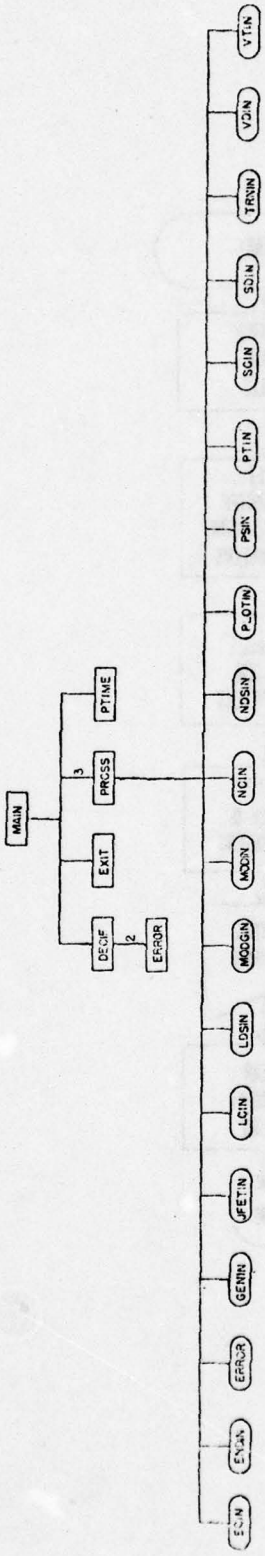
Machine dependent code has been clearly identified in both the program listing and narrative descriptions in order to ease the adaptation of NCAP to various computer systems.

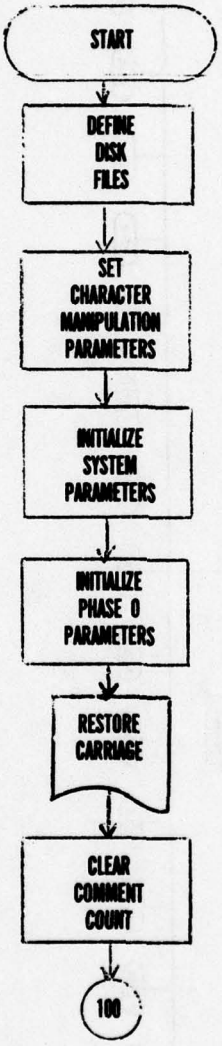
Functional flow diagrams for each subprogram have been

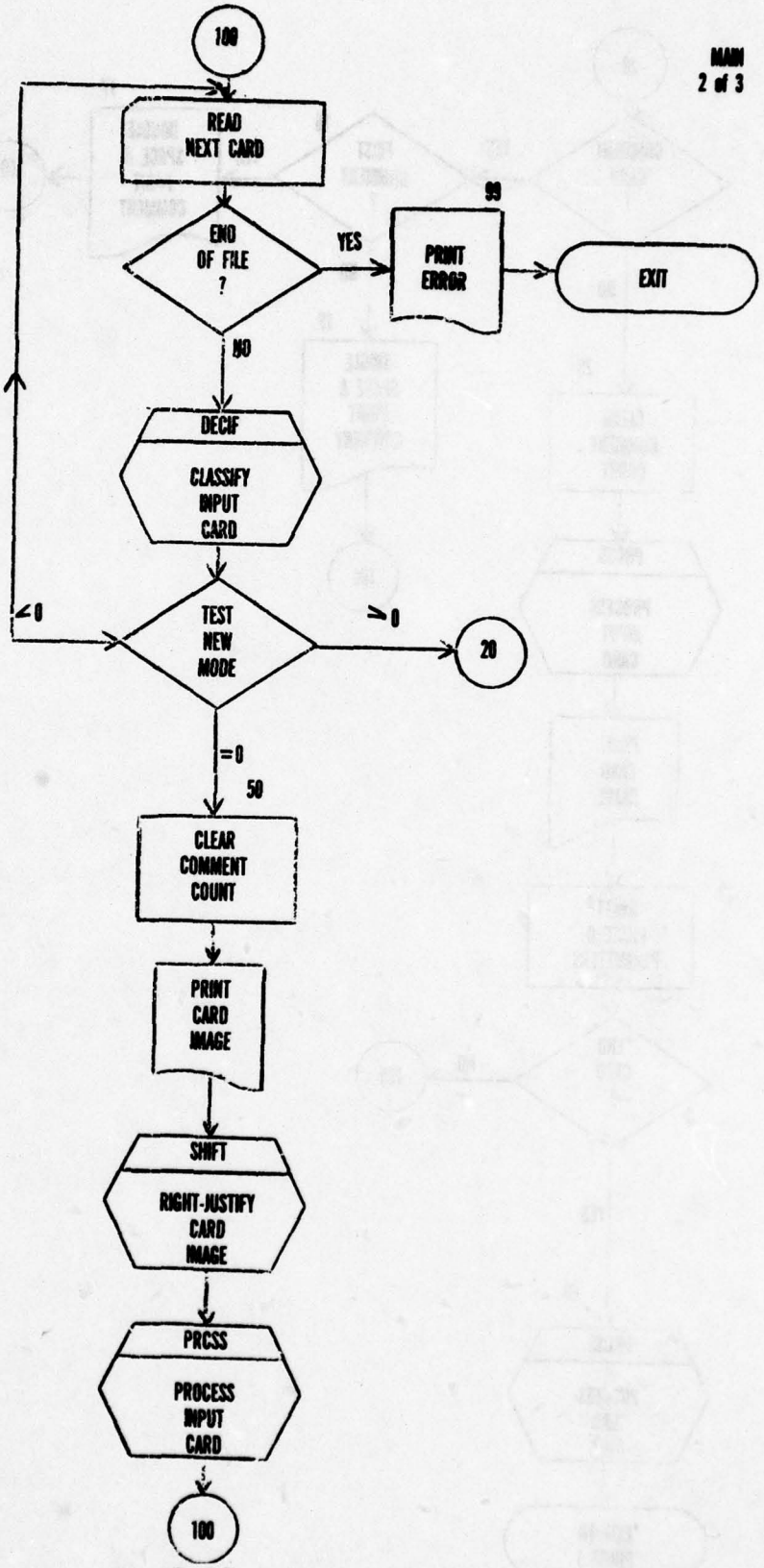
included to provide an insight into the general flow of logic through the program. They are not intended to describe the analytical techniques employed in the program or to depict the program code in detail. Individual symbols of the flow diagrams generally represent several lines of program code or complete code segments. Wherever possible, diagram symbols have been labelled with appropriate statement numbers from the program.

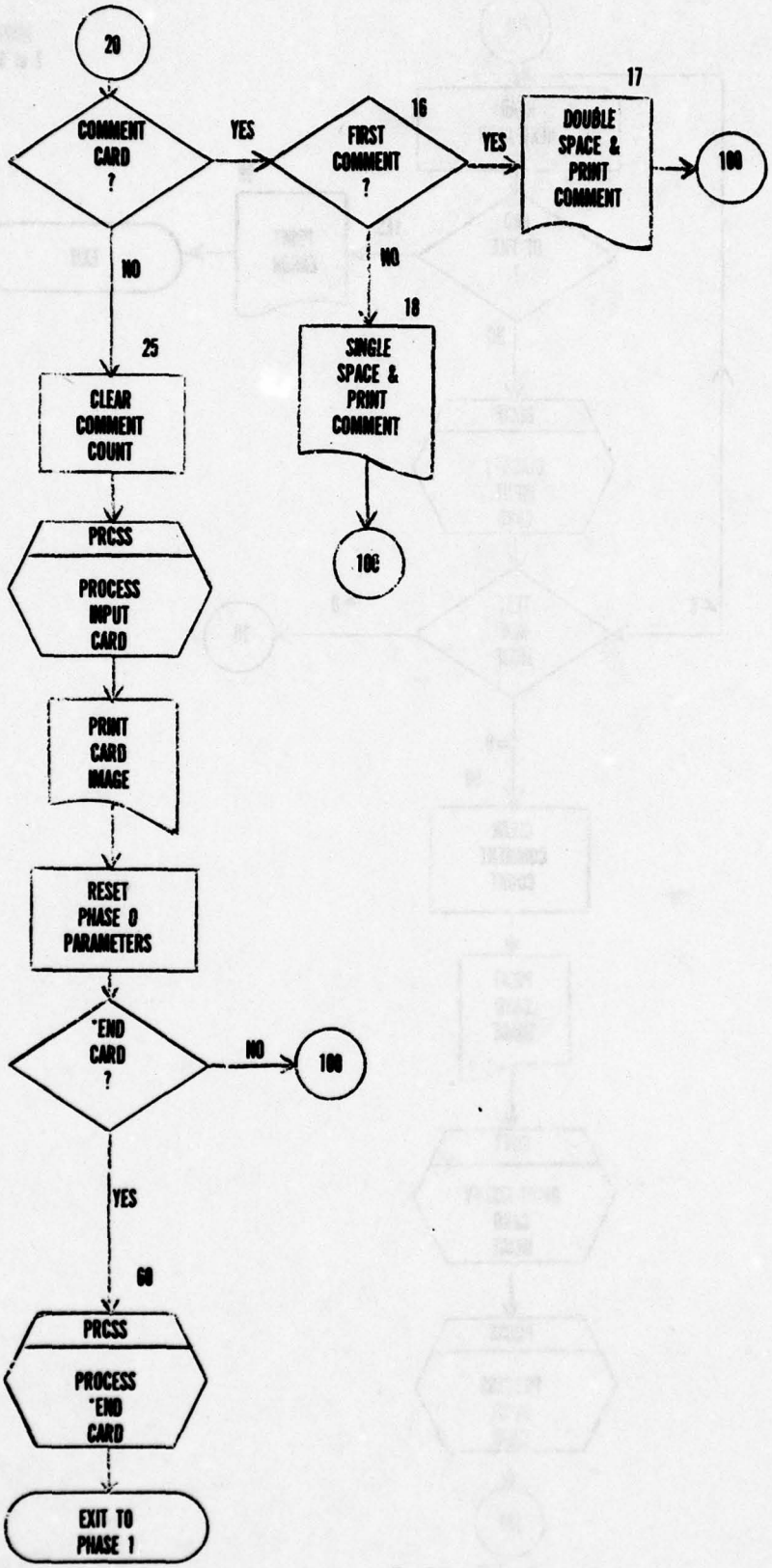
MAIN

ALPHA
DATOUT
DECIF
DECIM
DRVOUT
ECIN
ENDIN
GENIN
GENOUT
INTEG
JFETIN
LCIN
LDSIN
MODGIN
MODIN
NCIN
NDSIN
NODEIN
PACKL
PACKR
PLOTIN
PRCSS
PSIN
PTIN
SCAN
SCIN
SHIFT
TRNIN
VDIN
VTIN


MAIN
DEOF
EXIT
PROSS
PTIME
ERROR
ESIN
ENSIN
ERROR
GENIN
JPETIN
LGIN
LOSIN
MODGIN
MOOIN
NCIN
NDSIN
P_OTIN
PSIN
PTIN
SGIN
SOIN
TRSIN
VOIN
VTIN


START
DEFINE DISK FILES
SET CHARACTER MANIPULATION PARAMETERS
INITIALIZE SYSTEM PARAMETERS
INITIALIZE PHASE 0 PARAMETERS
RESTORE CARRIAGE
CLEAR COMMENT COUNT
100


100
READ
NEXT CARD
END
OF FILE
?
YES
99
PRINT
ERROR
EXIT
NO
DECIF
CLASSIFY
INPUT
CARD
TEST
NEW
MODE
<0
>0
20
=0
50
CLEAR
COMMENT
COUNT
PRINT
CARD
IMAGE
SHIFT
RIGHT-JUSTIFY
CARD
IMAGE
PRCSS
PROCESS
INPUT
CARD
100


20
COMMENT CARD ?
YES
16
FIRST COMMENT ?
YES
17
DOUBLE SPACE & PRINT COMMENT
100
NO
NO
18
SINGLE SPACE & PRINT COMMENT
10C
25
CLEAR COMMENT COUNT
PRCSS
PROCESS INPUT CARD
PRINT CARD IMAGE
RESET PHASE 0 PARAMETERS
*END CARD ?
NO
100
YES
60
PRCSS
PROCESS *END CARD
EXIT TO PHASE 1

NAME: MAIN

TYPE: MAIN PROGRAM

GENERAL PURPOSE:

Disk file definition, system initialization, control of input phase

VARIABLES:

BUFF = I/O buffer for disk records

CARD = Input buffer for card image. One character per word, alpha representation, left-justified

DATREC = Next available location on data file (20)

DRVREC = Driver file (21) address pointer

IMAGE = Print buffer for input card images

INP = Logical unit number of card reader

IOUT = Logical unit number of line printer

JAPT = Input error switch: off = 0, on = 1

LNGTH = Length of data record in words

MAXIT = Total number of frequency sweep iterations

MDESV = Mode of previous *-card

MISC = Driver record parameters unique to each mode

MODE = Identifies circuit element, device model, or NCAP function

1 = Drive file header

2 = Comment or no-op

3 = Generator

4 = Linear Components

5 = Nonlinear Components

6 = Vacuum Diode

7 = Vacuum Triode

8 = Vacuum Pentode

9 = Pipolar Junction Transistor

10 = Semiconductor Diode

11 = Not used

12 = Frequency Sweep

13 = Start Circuit

14 = End Circuit

15 = Not used

16 = End

17 = Print Select

18 = Modify

19 = Not used

20 = Junction Field Effect Transistor

21 = Generator Modification

22 = Plot

23 = Not used

24 = Linear Component Sweep

25 = Linear Dependent Source

26 = Nonlinear Dependent Source

MODFY = Modify switch: 0 = Modify not in effect

1 = Modify in effect

NAPT = Error switch for node card:

0 = Node card present

1 = Node card missing

NBCHAR = Number of bits per character

NBWORD = Number of bits per word

NCHAR = Number of characters per word

NCOM = Number of successive comment cards

NCURV = Number of curves defined by circuit analysis

NEW = Identifies input card being processed:

-1 = Unidentified card, card out of order

Ø = Parameter card

1 = Not used

2 = Comment card

>2 = *-card

NFR = Number of frequency values for abcissa of plot

NIT = Frequency sweep iteration counter

NLIST = Print select on/off switch; Ø = on, 1 = off

NPLOT = Number of plot specifications in input deck

NVEC = Number of t.f. values for ordinate of plot

NXT = Counter for data values being input

ORDER = Total number of nodes in circuit; Order of admittance matrix

SCREC = Location of *START CIRCUIT driver record

STADD = Data file (2Ø) address pointer

TIME = Time elapsed in single program segment

TSTART = Start time of single program segment

TSTOP = Stop time of single program segment

TTOTAL = Total time elapsed all executions single

program segment

TYPE = Identifies type of parameter card:

1 = *-card

2 = Data values only

3 = Impedance

4 = Frequency

5 = Order

6 = Node

7 = Amplitude

8 = Resistor

9 = Capacitor

10 = Inductor

11 = Constants

12 = BP

13 = CP

14 = AC

15 = Label

16 = FP

17 = RB

18 = VC

19 = CC

20 = VV

21 = CV

22 = Off

23 = On

SUBROUTINES CALLED:

DECIF PRCSS PTIME RANSIZE RSHIFT SHIFT

CALLING PROGRAMS:

NONE

DESCRIPTION:

NCAP's main program is the primary routine of Phase 0. It performs initialization of system parameters and reads the input deck. System initialization begins with the definition of NCAP's disk files. Because different Fortran compilers employ various methods for specifying random access files, this function is considered machine dependent. Therefore the structure and method of accessing NCAP's disk files are presented here to enable the reader to translate the particular random access I/O requirements of NCAP into appropriate file definition statements.

The data file, logical unit 20, contains the NCAP input data and all internally generated data and results such as device parameters, admittance matrices, and transfer function vectors. The data file is structured as an unlimited number of contiguous one - word physical records, each of which is addressable according to its relative location in the file. A logical record, such as a device data record or a transfer function vector, consists of an arbitrary number of physical records and is accessed by the first word of the record and the record length. The integer variables STADD and DATREC are used as data file record pointers. Subroutine DATAWR causes data to be written from core storage to file 20, while subroutine DATARD

causes data to be read from file 20 to core storage.

The driver file, logical unit 21, is a mapping of the * - input cards to disk storage. It contains the information required to define the circuit and the circuit analyses to the system, and is used to control the operation of the various NCAP phases. The driver file is initially created during Phase 0 as input cards are processed, and is modified or updated as necessary during subsequent phases. Each * - card in the input deck maps to one or more driver file records, which originate from the first ten words of global common. The first three words of each driver record, MODE, STADD, and LNGTH, identify the function and define the boundaries of its associated data record on file 20. The additional words MISC(1) through MISC(7) contain parameters and pointers unique to each NCAP function. The driver file is accessed by subroutines DRIVWR and DRIVRD. Individual driver records are addressed by the record number stored in DRVREC.

File 22 contains frequency data used as the abscissa for plotting NCAP output. Each record contains ten frequency values, one for each of the ten possible plot specifications in the NCAP input deck. The number of records in the file is maintained in the integer word NFR and depends upon the number of circuit analyses performed in the NCAP run. File 22 is accessed directly by random file read/write statements.

File 23 contains the transfer function values used as the ordinate for plotting NCAP output. Each record contains twenty words and stores one complex (two- word) transfer function value

for each of the ten possible plot specifications in the NCAP input deck. The number of records in the file is maintained in the integer word NVEC and depends upon the number of circuit analysis performed in the NCAP run. File 23 is accessed directly by random file read/write statements.

File 24 contains the transfer function table created in Phase 3 by subroutine CONTRL. It is made up of the lower-order transfer function vectors required by the current generator subroutines, and consists of a large number of contiguous one-word physical records, each of which is addressable. Each logical record stores one complex transfer function vector whose length depends upon the number of nodes in the circuit. Transfer function vectors are accessed according to LOCT, the address of the first word of the record, and LT, the length of the record. Transfer function values at a particular node are accessed by the first word of the vector plus a displacement factor derived from the node number. Subroutine TFWR and TFRD perform input/output for file 24.

File 25 is used to store the current vector calculated for one frequency combination in Phase 3. The file contains one logical record, the length of which is determined by the number of nodes in the circuit. Current vector values for each node of the circuit are individually addressable. Subroutines TFWR and TFRD perform input/output for file 25.

The system initialization portion of Phase 0 begins with the definition of NCAP'S machine dependent character manipulation parameters which are transmitted to the subroutines which use

them through the labelled common area WORDSZ. The integer variables NCHAR (number of characters per word), NBCHAR (number of bits per character), and NBWORD (number of bits per word) describe precisely the word length of the host computer and its method of representing alphanumeric characters internally. The alphanumeric left-justified blank, LBLANK, is defined in a DATA statement. The right-justified blank character, RBLANK, is calculated in the main program by a call to subroutine SHIFT.

Next the logical unit numbers of the primary input and output devices, usually the card reader and line printer, are stored in INP and IOUT respectively. The frequency sweep parameters NIT and MAXIT, the plot parameters NCURV, NFR, and NVEC, and the print select and plot switches NLIST and NPLOT are initialized in the main program and updated and tested during the execution of subsequent phases.

The input phase begins with the initialization of Phase 0 pointers, switches, and data areas. The data buffer BUFF(I), I=1, 250 is cleared and the driver file record number DRVREC is set to 2, leaving an open record for the driver file header which is generated internally at the conclusion of Phase 0. DATREC, the next available storage location on data file 20, is initialized to 1, as is the data file address pointer STADD. The input error switch is turned off by JABT = 0, while the node card error switch is initially turned on by NABT = 1. The data value and comment card counts NXT and NCOM are cleared and the Start Circuit driver record number SCREC is set to zero.

Since the exact content of the NCAP input deck is not known

in advance, input processing is not merely a matter of reading and storing data in a prearranged fashion. Instead the subroutines of Phase Ø act in concert to interpret the input cards, translating them into appropriate driver and data records for use by subsequent NCAP phases.

The input deck consists of a series of *-control card sequences which define circuit topology, circuit excitation, linear and nonlinear elements and models, solution modification, and output. Each such card sequence begins with an *-control card defining its function, followed by parameter cards that supply additional information and/or precise values for that function.

Phase Ø is driven primarily by the *-control cards. Whenever an *-control card is encountered in the input stream, the previous function is closed by writing its driver and data records to disk, and a new function is opened. Parameter (nonasterisk) cards are passed along to a subroutine unique to the function defined by the previous *-card. There the parameter card is processed according to an anticipated format and its data merged with any previous data for that function. Input continues in this fashion until the *END card is processed, at which time program control transfers to PHASE1.

Input processing is controlled by NCAP'S main program and begins by initializing the function identifier MODE to 1, indicating that no input card sequences have yet been processed. A card is read in left-justified alphanumeric format, one character per word, into the integer array CARD. If an

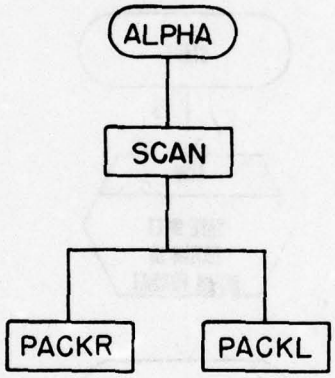
unanticipated end-of-file is encountered, execution terminates with a CALL EXIT. Otherwise the card image is transmitted to subroutine DECIF where it is examined and assigned a functional integer code which is returned to the main program in NEW. Upon return from DECIF, NEW<0 indicates an unidentified card which is not processed further, and after printing an appropriate error message program control transfers to read the next card. Otherwise the card is either a comment (**), an *-control card, or a parameter (non-asterisk) card. Comment cards, identified by NEW=2, are printed on the line printer but otherwise ignored by Phase 0.

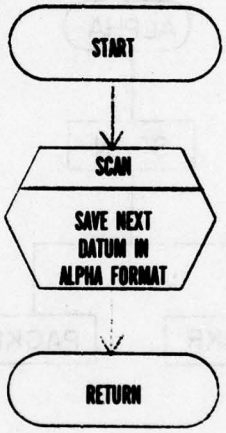
When an *-control card is encountered in the input deck, subroutine DECIF returns a value of NEW>2 to the main program. This causes the old function identified by MODE to be closed and the new function identified by NEW to be opened. Control passes from the main program, through subroutine PRCSS, to an appropriate input processing subroutine which closes the previous *-function by writing its driver and data records to disk, and printing any error messages associated with it. Upon return to the main program the new *-card is printed, the mode of the previous function is saved in MDESV, and the modify switch MODFY is turned off. The new function is opened by setting MODE=NEW and the data value counter, driver record parameters, and data buffer are reinitialized.

Parameter card images (NEW=0) are output on the line printer, and then transmitted through subroutine PRCSS to an appropriate input processing subroutine according to the value of

MODE. There the card is reformatted and interpreted according to the value of TYPE. Upon return to the main program, control transfers to read the next card.

Input continues until the *END (MODE=16) card is encountered. At this point the input processor ENDIN causes program control to transfer to PHASE1 if no errors were detected in the input. Otherwise NCAP execution terminates.


ALPHA
SCAN
PACKR
PACKL


START
SCAN
SAVE NEXT DATUM IN ALPHA FORMAT
RETURN

NAME· ALPHA

TYPE· SUBROUTINE

GENERAL PURPOSE·

Decodes and formats an alphabetic input value by converting the characters stored in CARD(IPOS) through CARD(J-1) to an alphanumeric value returned in STRING(1)

VARIABLES:

CARD = Right-justified alphanumeric card image
IPOS = Position of first character to be formatted
ISW = End of data switch; Off = 0, On = 1
J = Position of last character to be formatted+1
STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED·

SCAN

CALLING PROGRAMS·

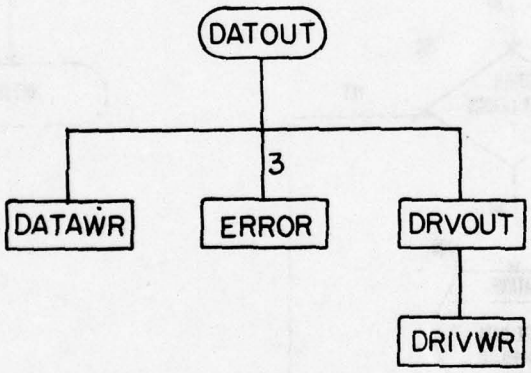
GENIN	JFETIN	LCIN	LDSIN	MODGIN
NCIN	NDSIN	NODEIN	PLOTIN	PSIN
SDIN				

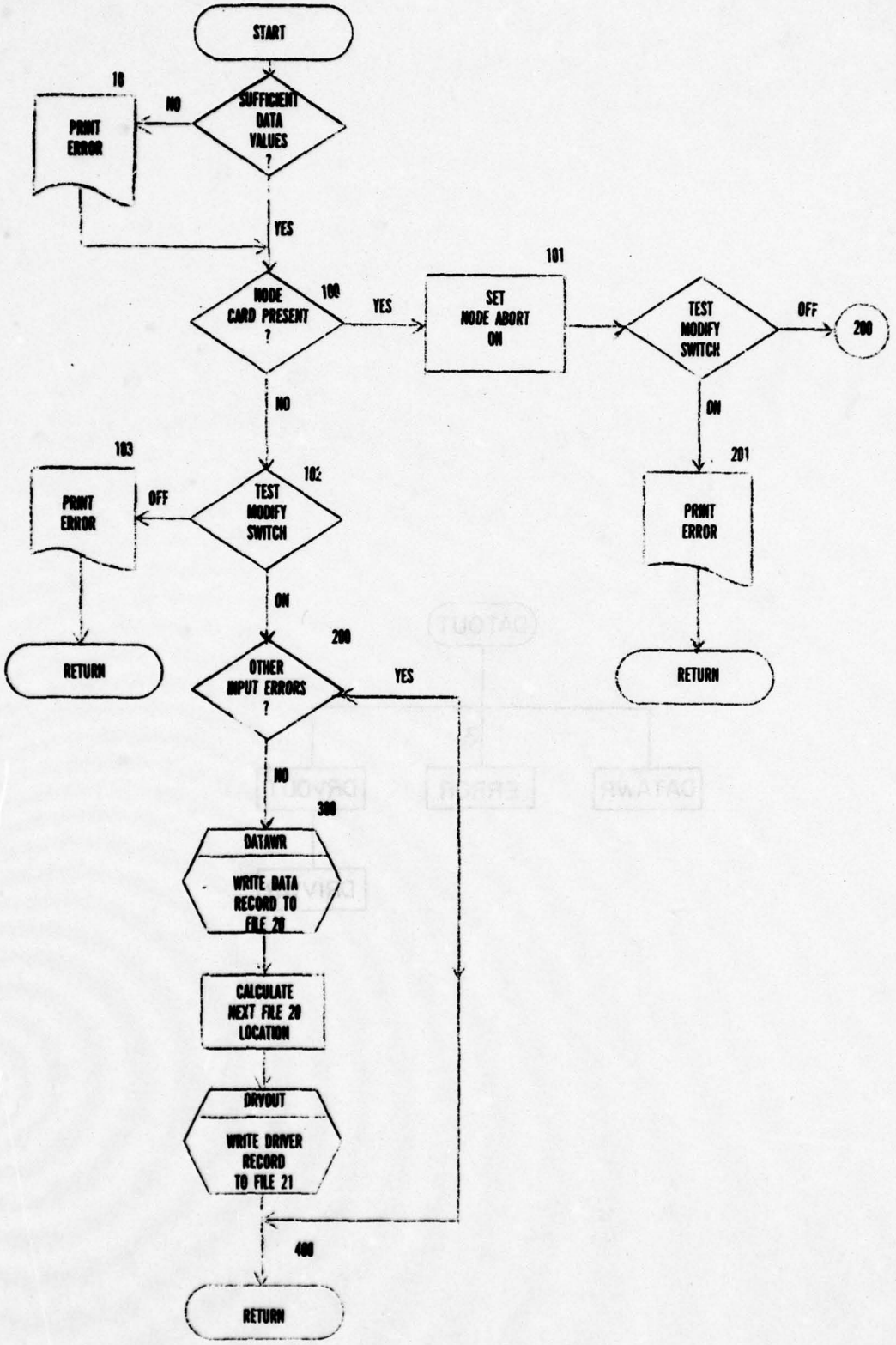
DESCRIPTION·

Subroutine ALPHA decodes and formats a free-form alphabetic input value. Input to the routine is the right-justified alphanumeric card image stored in the integer array CARD and the character position IPOS at which the datum begins.

The contiguous characters CARD(IPOS) through CARD(J-1) which comprise the alphabetic input value are extracted from the card image and converted to a left-justified alphanumeric string by subroutine SCAN. The resulting alphabetic datum stored in the integer STRING(1) is returned to the calling program without further processing.

Additional arguments returned by subroutine ALPHA are the updated character position IPOS, which indicates the first character position of the next value on the card, and the end-of-data switch ISW.


DATOUT
3
DATAWR
ERROR
DRVOUT
DRIVWR


START
SUFFICIENT DATA VALUES ?
NO
10
PRINT ERROR
YES
MODE CARD PRESENT ?
100
YES
101
SET MODE ABORT ON
TEST MODIFY SWITCH
OFF
200
ON
201
PRINT ERROR
RETURN
NO
103
PRINT ERROR
OFF
TEST MODIFY SWITCH
102
ON
RETURN
OTHER INPUT ERRORS ?
200
YES
NO
300
DATAWR
WRITE DATA RECORD TO FILE 20
CALCULATE NEXT FILE 20 LOCATION
DRVOUT
WRITE DRIVER RECORD TO FILE 21
400
RETURN

NAME: DATOUT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Writes input data to file 20 and driver records to file 21. Output is by-passed if input errors exist. Clears I/O buffer.

VARIABLES:

J = Number of parameters in the data record

N = Length of data record to be output (in words)

SUBROUTINES CALLED:

DATAWR DRVOUT ERROR

CALLING PROGRAMS:

PTIN SCIN SDIN TRNIN

VDIN VTIN JFETIN

DESCRIPTION:

Subroutine DATOUT handles the output of data and driver records for many of the Phase 0 input processing routines, closing the function defined by an *-card sequence. Arguments transmitted to the subroutine are J, the number of parameters

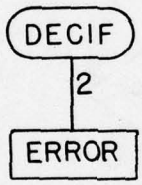
anticipated and N, the length of the data record in words.

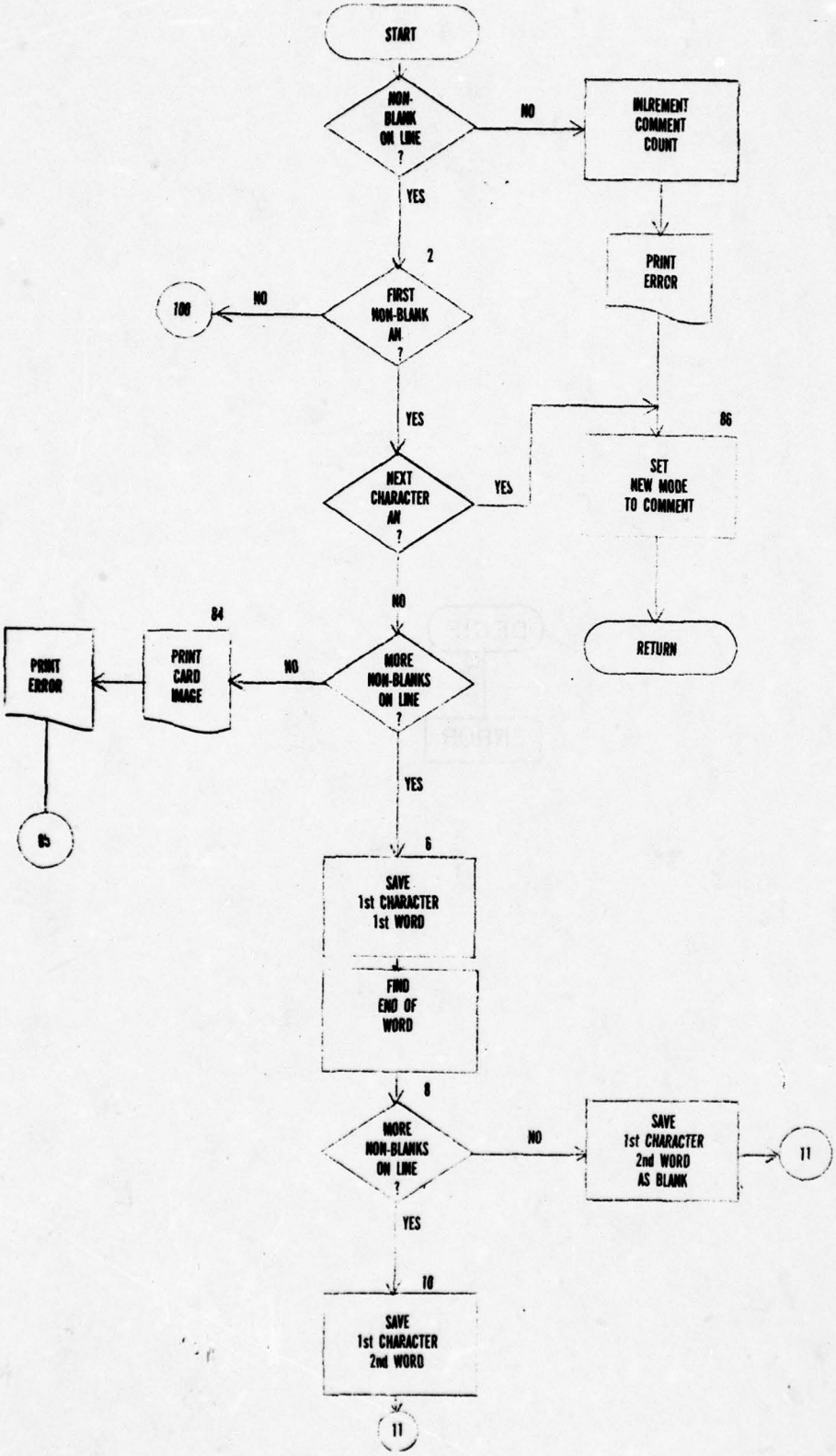
First the input data is tested for errors. Subroutine DATOUT exits with an appropriate error message if any of the following input errors are detected:

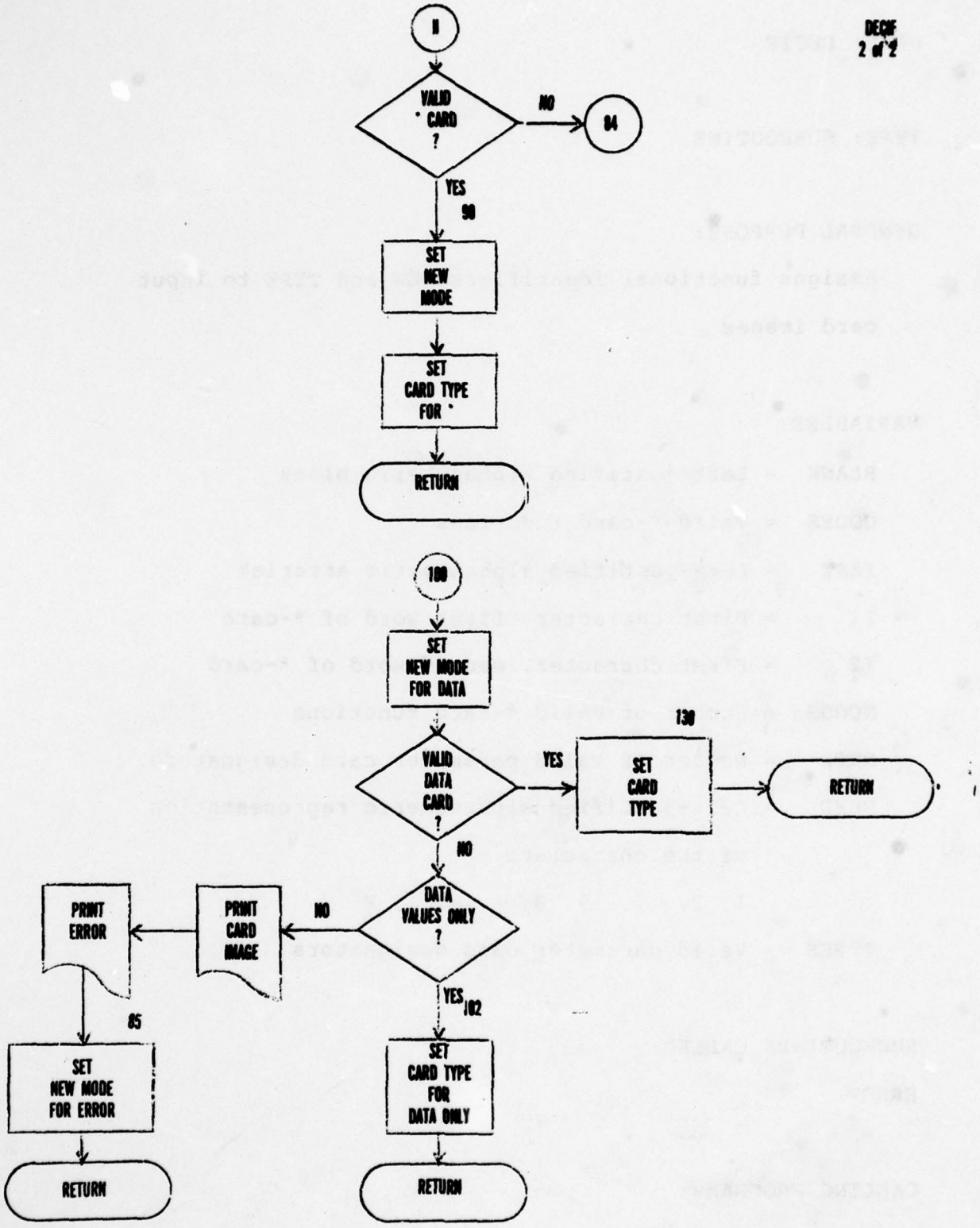
1) The parameter count NXT is less than the anticipated number of data values
2) A node card was encountered in a modify card sequence (NABT=0 and MODFY=1)
3) The node card was missing from a circuit element definition (NABT=1 and MODFY=Ø)
4) The input error switch is on (JABT=1)

If no input errors are detected, the data record is written from the common data buffer BUFF to file 2Ø by subroutine DATAWR. The data file record number DATREC is updated to point beyond the new data record by DATREC = DATREC + N.

Subroutine DRVOUT is called to complete the driver record definition and to write it to file 21, closing the function. Subroutine DATOUT then returns to the calling program.


DECIF
2
ERROR


START
NON-BLANK ON LINE ?
NO
INCREMENT COMMENT COUNT
YES
PRINT ERROR
2
FIRST NON-BLANK AN ?
NO
100
YES
86
SET NEW MODE TO COMMENT
NEXT CHARACTER AN ?
YES
NO
RETURN
84
PRINT ERROR
PRINT CARD IMAGE
NO
MORE NON-BLANKS ON LINE ?
YES
85
6
SAVE 1st CHARACTER 1st WORD
FIND END OF WORD
8
MORE NON-BLANKS ON LINE ?
NO
SAVE 1st CHARACTER 2nd WORD AS BLANK
11
YES
10
SAVE 1st CHARACTER 2nd WORD
11


11
VALID ' CARD ?
NO
84
YES
90
SET NEW MODE
SET CARD TYPE FOR '
RETURN
100
SET NEW MODE FOR DATA
VALID DATA CARD ?
YES
130
SET CARD TYPE
RETURN
NO
DATA VALUES ONLY ?
NO
PRINT CARD IMAGE
PRINT ERROR
YES
102
85
SET CARD TYPE FOR DATA ONLY
SET NEW MODE FOR ERROR
RETURN
RETURN

NAME: DECIF

TYPE: SUBROUTINE

GENERAL PURPOSE:

Assigns functional identifiers NEW and TYPE to input card images

VARIABLES:

BLANK = Left-justified alphanumeric blank

CODES = Valid *-card functions

IAST = Left-justified alphanumeric asterisk

I1 = First character, first word of *-card

I2 = First character, second word of *-card

NCODS = Number of valid *-card functions

NTPS = Number of valid parameter card designators

NUMS = Left-justified alphanumeric representation of the characters

1, 2, ... 9, 0, +, -, ., E

TYPES = Valid parameter card designators

SUBROUTINES CALLED:

ERROR

CALLING PROGRAMS:

MAIN

DESCRIPTION:

Subroutine DECIF examines the card image transmitted to it in the integer array CARD and assigns NEW and TYPE codes according to its contents. The card image is stored in left-justified alphanumeric format, one character per word in the integer array CARD.

Although the system cannot anticipate the exact make-up of an input deck, it does recognize five basic card types:

1) An *-control card containing an * followed by one or more keywords. Such cards are classified by TYPE=1 and NEW=3,4,...26, depending on the keywords.
2) A parameter card containing an identifying keyword. These cards are classified by NEW=0 and TYPE= 3,4,...23 depending on the keyword.
3) A parameter card containing only numeric values and classified by NEW=0 and TYPE=2.
4) A comment card containing ** in the first two non-blank character positions. Comments are classified by NEW=2.
5) An unidentifiable card constituting an error and classified by NEW=-1.

The function of subroutine DECIF is to determine which of these card types applies to the given card image and to assign corresponding NEW and TYPE codes which serve as the basis for further processing by Phase 0. A complete tabulation of NEW and TYPE codes is presented below:

NEW (MODE)	CODES (Character Pair)	DESCRIPTION
-1	None	Error Condition
Ø	None	Parameter Card
1	None	Driver File Header (Internally Created)
2	None	Comment
3	G_	Generator
4	LC	Linear Components
5	NC	Nonlinear Components
6	VD	Vacuum Diode
7	VT	Vacuum Triode
8	VP	Vacuum Pentode
9	T_	Transistor
1Ø	SD	Semiconductor Diode
11	XX	Not Used
12	XX	Frequency Sweep (Internally Created)
13	SC	Start Circuit
14	EC	End Circuit
15	XX	Not Used
16	E_	End
17	PS	Print Select
18	M_	Modify
19	XX	Not Used

20	J	Junction Field Effect Transistor
21	XX	Generator Modification (Internally Created)
22	P_	Plot
23	XX	Not Used
24	XX	Linear Component Sweep (Internally Created)
25	LD	Linear Dependent Source
26	ND	Nonlinear Dependent Source

TYPE	TYPES *(Character Pair)*	DESCRIPTION
1	None	*-Control Card
2	None	Parameter Card With Only Numeric Values
3	IM	Impedance
4	FR	Frequency
5	OR	Orders for Print Select or Plot
6	NC	Nodes for Print Select or Plot
7	AM	Amplitude
8	R_	Resistor
9	C_	Capacitor
10	L_	Inductor

11	CO	Constants for JFET
12	BP	B-Parameters for JFET
13	CP	C-Parameters for JFET
14	AC	Analytic Parameters for JFET
15	LA	Label for Plot
16	FB	Forward Bias
17	RB	Reverse Bias
18	VC	Voltage Controlled Current Source
19	CC	Current Controlled Current Source
20	VV	Voltage Controlled Voltage Source
21	CV	Current Controlled Voltage Source
22	OF	Print Select Off
23	ON	Print Select On

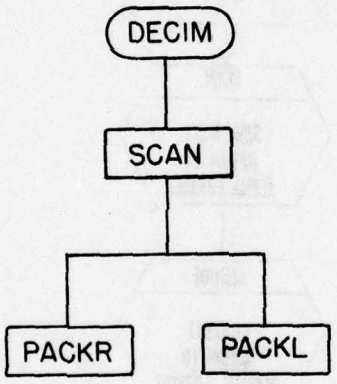
Because the NCAP user expresses his input in free-form, there is no requirement for card input to begin in column 1 or that data values appear in particular card columns. A data value is defined as a string of contiguous non-blank alphanumeric characters, and although the order in which data are placed on cards is critical, the spacing between data values is arbitrary. Therefore subroutine DECIF must isolate the individual data values on the card and then classify the card according to those values.

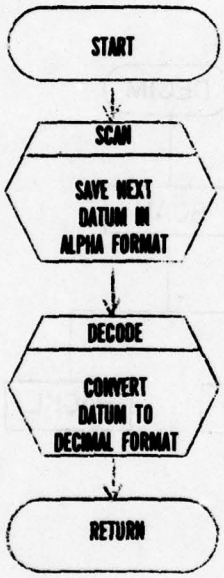
Classification begins by determining the presence or absence of an * in the first non-blank character position. Starting at CARD(1) the image is scanned column-by-column to locate the first non-blank character. If no non-blanks are found, the card contains no data and subroutine DECIF exits to the main program treating the blank card as a comment. Otherwise the first non-blank located in CARD(I) is tested for the presence of an *. If an * is found, the card is either an *-control card or a comment card. If CARD(I+1) is also an asterisk, a comment is assumed and classified by NEW=2. After incrementing the comment count NCOM, subroutine DECIF exits to the main program.

If CARD(I+1) is not an asterisk, an *-control card is assumed. The first character of each of the next two words is extracted and compared against the table of valid character pairs stored in CODES(J), J=1, NCODS. If a match is found, the card is classified by TYPE=1 and NEW=J+2. If no match is found, the unidentifiable card is classified by NEW=-1 and exit occurs after an appropriate error message is printed.

In the event that the first non-blank character CARD(I) is not an asterisk, a parameter card is assumed and classified by NEW=0. Such cards are further examined by comparing the first two characters of the first word on the card against a table of valid character pairs stored in TYPES(J), J=1, NTPS. If a match is found, the card is classified by TYPE=J+2 and exit occurs. Otherwise the first non-blank character is compared against the table of valid numeric characters stored in NUMS(J), J=1, 14. No

match indicates an unidentifiable card which is assigned NEW=-1. If the character is a valid numeric, the card is assumed to contain only numeric data values and is further classified by TYPE=2. In either event return is made to the main program where further processing of the card image takes place.


DECIM
SCAN
PACKR
PACKL


START
SCAN
SAVE NEXT DATUM IN ALPHA FORMAT
DECODE
CONVERT DATUM TO DECIMAL FORMAT
RETURN

NAME: DECIM

TYPE: SUBROUTINE

GENERAL PURPOSE:

Decodes and formats a decimal input value by converting the characters stored in CARD(IPOS) through CARD(J-1) to a floating point number returned in X

VARIABLES:

CARD	= Right-justified alphanumeric card image
IPOS	= Position of first character to be formatted
ISW	= End of data switch: Off = 0, On = 1
J	= Position of last character to be formatted+1
STRING	= Alphanumeric string representation of input value
X	= Floating point value returned

SUBROUTINES CALLED:

SCAN

CALLING PROGRAMS:

CFNIN	JFETIN	LCIN	LDSIN	MODCIN
MCIN	MDSIN	PTIN	SDIN	TRNIN
VDIN	VTIN			

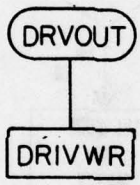
DESCRIPTION·

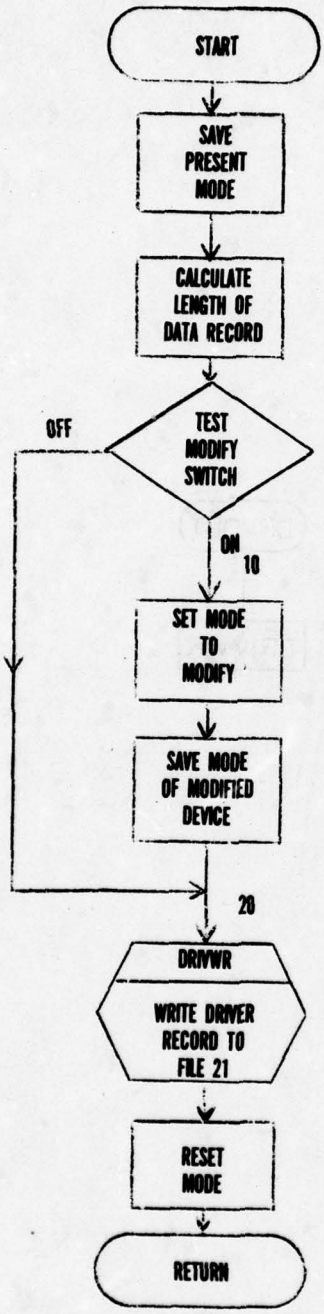
Subroutine DECIM decodes and formats a free-form decimal input value. Input to the routine is the right-justified alphanumeric card image stored in the integer array CARD and the character position IPOS at which the datum begins.

The contiguous characters CARD(IPOS) through CARD(J-1) which comprise the decimal input value are extracted from the card image and converted to a right-justified alphanumeric string by subroutine SCAN. The character string returned in the integer array STRING is then converted from alphanumeric representation to a floating point value X by:

```
      DECODE (STRING, 1) X
    1 FORMAT (E12.0)
```

Arguments returned by subroutine DECIM are the decimal input value X, the updated character position IPOS which indicates the first character position of the next data value on the card, and the end-of-data switch ISW.


DRVOUT
DRIVWR


START
SAVE PRESENT MODE
CALCULATE LENGTH OF DATA RECORD
TEST MODIFY SWITCH
OFF
ON
10
SET MODE TO MODIFY
SAVE MODE OF MODIFIED DEVICE
20
DRVWR
WRITE DRIVER RECORD TO FILE 21
RESET MODE
RETURN

NAME: DRVOUT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Completes definition of driver parameters and writes driver record to file 21

VARIABLES:

JJ = Intermediate storage of MODE

SUBROUTINES CALLED:

DRIVWR

CALLING PROGRAMS:

DATOUT ECIN ENDIN LCIN LDSIN

NCIN PSIN SCIN

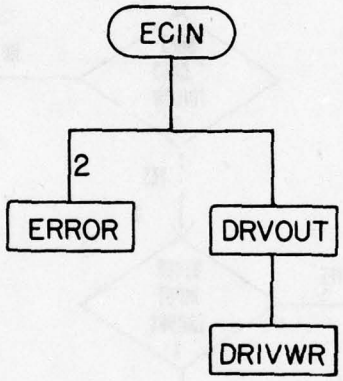
DESCRIPTION:

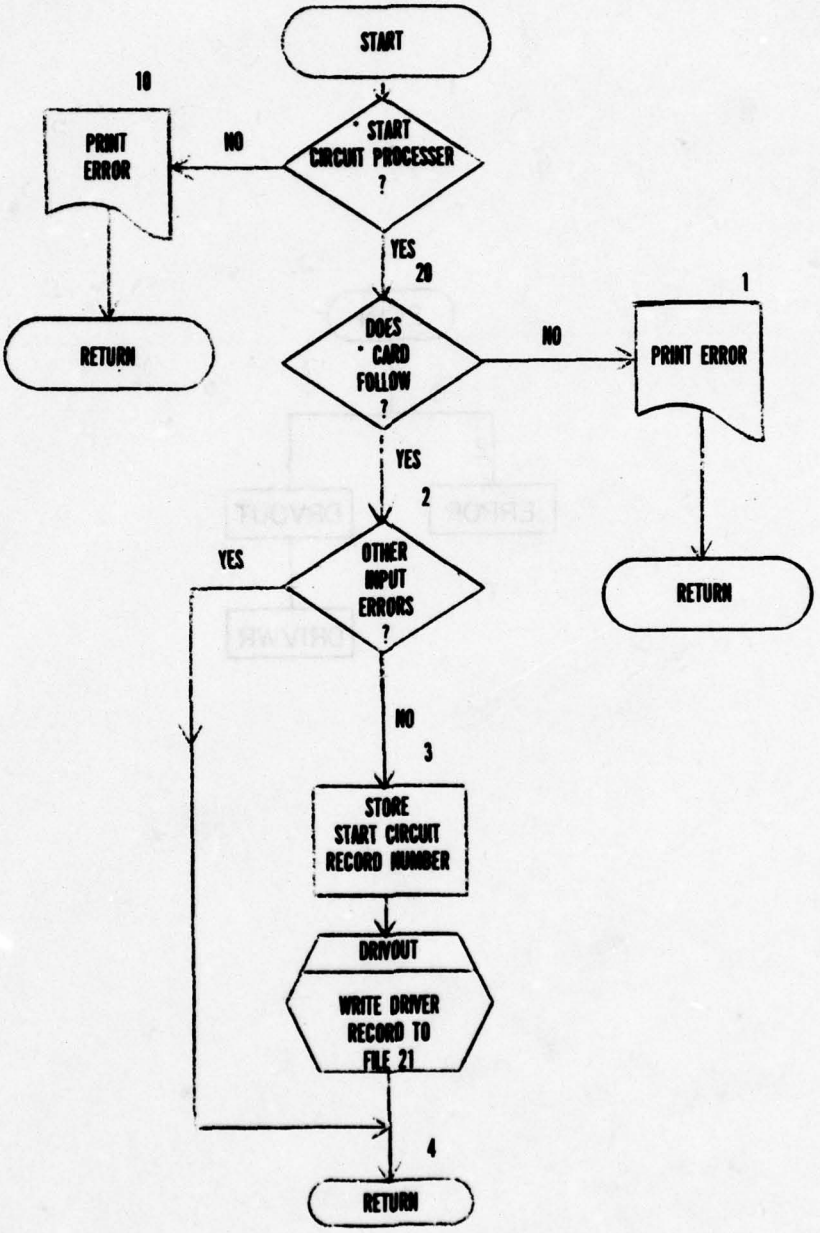
Subroutine DRVOUT completes the definition of driver record parameters for some of the Phase 0 input processing subroutines and writes the driver record to file 21.

The MODE of the present card sequence is saved in JJ and the data record length is calculated by LNGTH=DATREC-STADD. If the modify switch is off (MODFY=0), the driver record is written to file 21 from the first ten words of global common by subroutine DRIVWR.

A MODIFY card sequence carries the MODE of the device it modifies during input processing, but is written to the driver file as MODE=18. Therefore, if the card sequence being processed is a modify function (MODFY=1), MODE is set to 18 and the mode of the device being modified is taken from MDESV and saved in JJ before the driver record is written by subroutine DRIVWR.

Upon return from subroutine DRIVWR the record number for the next data record STADD is taken from DATREC, the mode of the present card sequence is reset from JJ, and subroutine DRVOUT returns to the calling program.


ECIN
2
ERROR
DRVOUT
DRIVWR


START
10
PRINT ERROR
NO
* START CIRCUIT PROCESSER ?
YES
20
RETURN
DOES * CARD FOLLOW ?
NO
1
PRINT ERROR
YES
2
YES
OTHER INPUT ERRORS ?
RETURN
NO
3
STORE START CIRCUIT RECORD NUMBER
DRYOUT
WRITE DRIVER RECORD TO FILE 21
4
RETURN

NAME: ECIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *END CIRCUIT card

VARIABLES:

NONE

SUBROUTINES CALLED:

ERROR

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

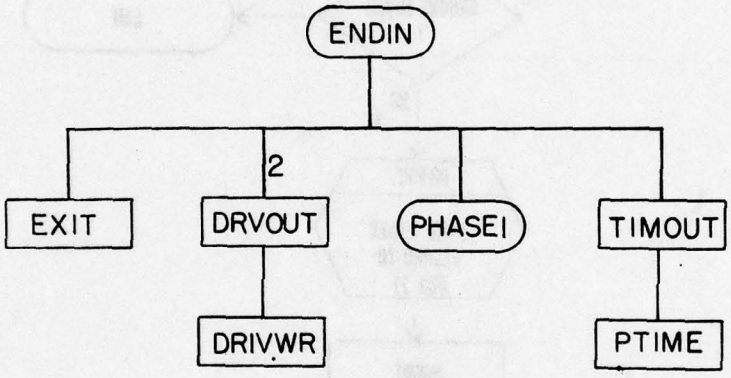
Subroutine ECIN is the input processor for the END CIRCUIT card. Its function is to translate the end circuit definition into an appropriate driver record for use by subsequent NCAP phases.

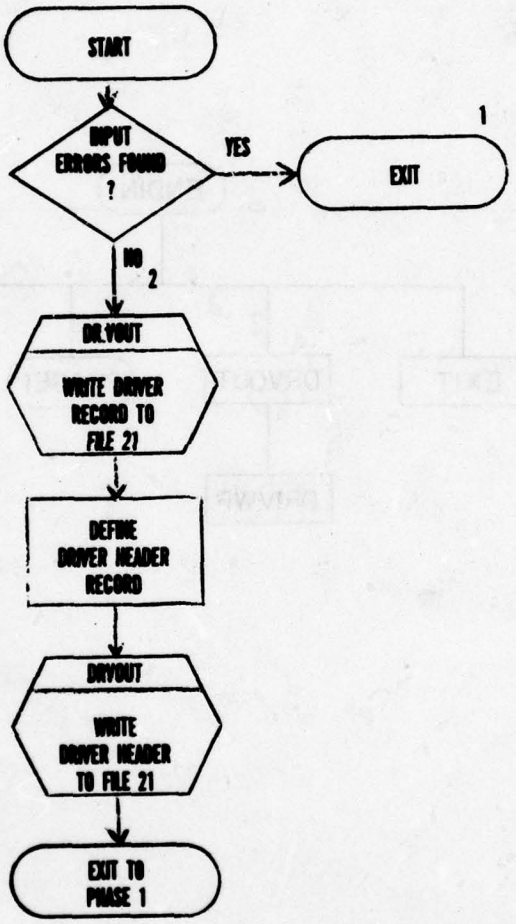
Because there are no data cards associated with the end circuit input, TYPE=1 is the only code recognized by subroutine ECIN. Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.

If the start circuit driver record number is undefined (SCREC=0) or the input error switch is on (JABT=1), subroutine ECIN exits with an appropriate error condition. Otherwise the *END CIRCUIT* driver record is written to file 21 from the first ten words of global common by subroutine DRVOUT.

The end circuit driver record is identified by MODE=14. Because there is no data associatd with an end circuit function, the driver parameters STADD and LNGTH are not applicable. The start circuit driver record number is stored in MISC(1). The other MISC driver parameters are not used.

After the driver record has been written to disk, subroutine ECIN returns to the calling program.


ENDIN
2
EXIT
DRVOUT
PHASEI
TIMOUT
DRIVWR
PTIME


START
INPUT ERRORS FOUND ?
YES
1
EXIT
NO
2
DR.VOUT
WRITE DRIVER RECORD TO FILE 21
DEFINE DRIVER HEADER RECORD
DRVOUT
WRITE DRIVER HEADER TO FILE 21
EXIT TO PHASE 1

NAME: ENDIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *END card, creates driver file header, closes Phase 0 processing

VARIABLES:

NONE

SUBROUTINES CALLED:

DRVOUT PHASE1 TIMOUT

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

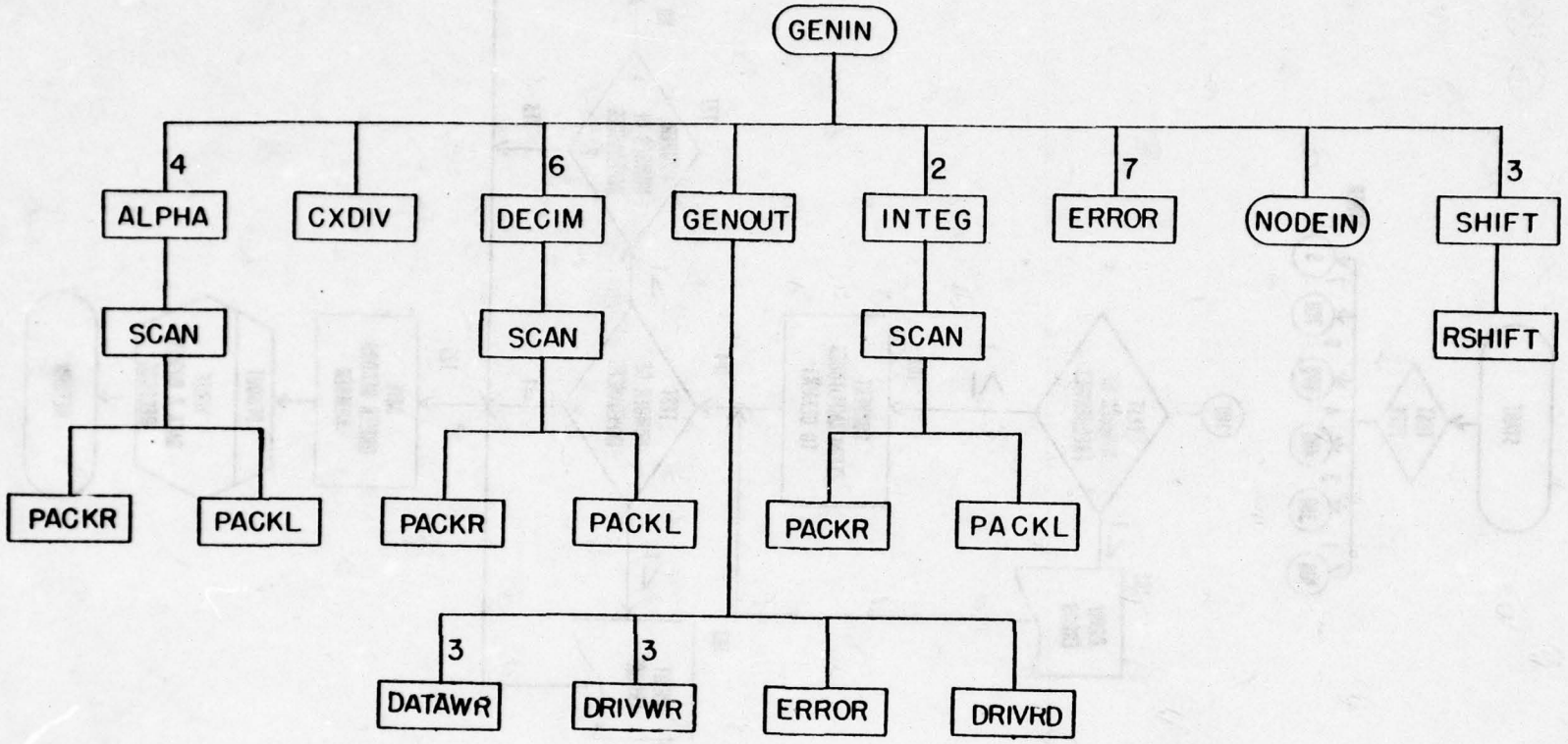
Subroutine ENDIN is the input processor for the END card. Its function is to translate the end definition into an appropriate driver record for use by subsequent NCAP phases and to close the Phase 0 processing.

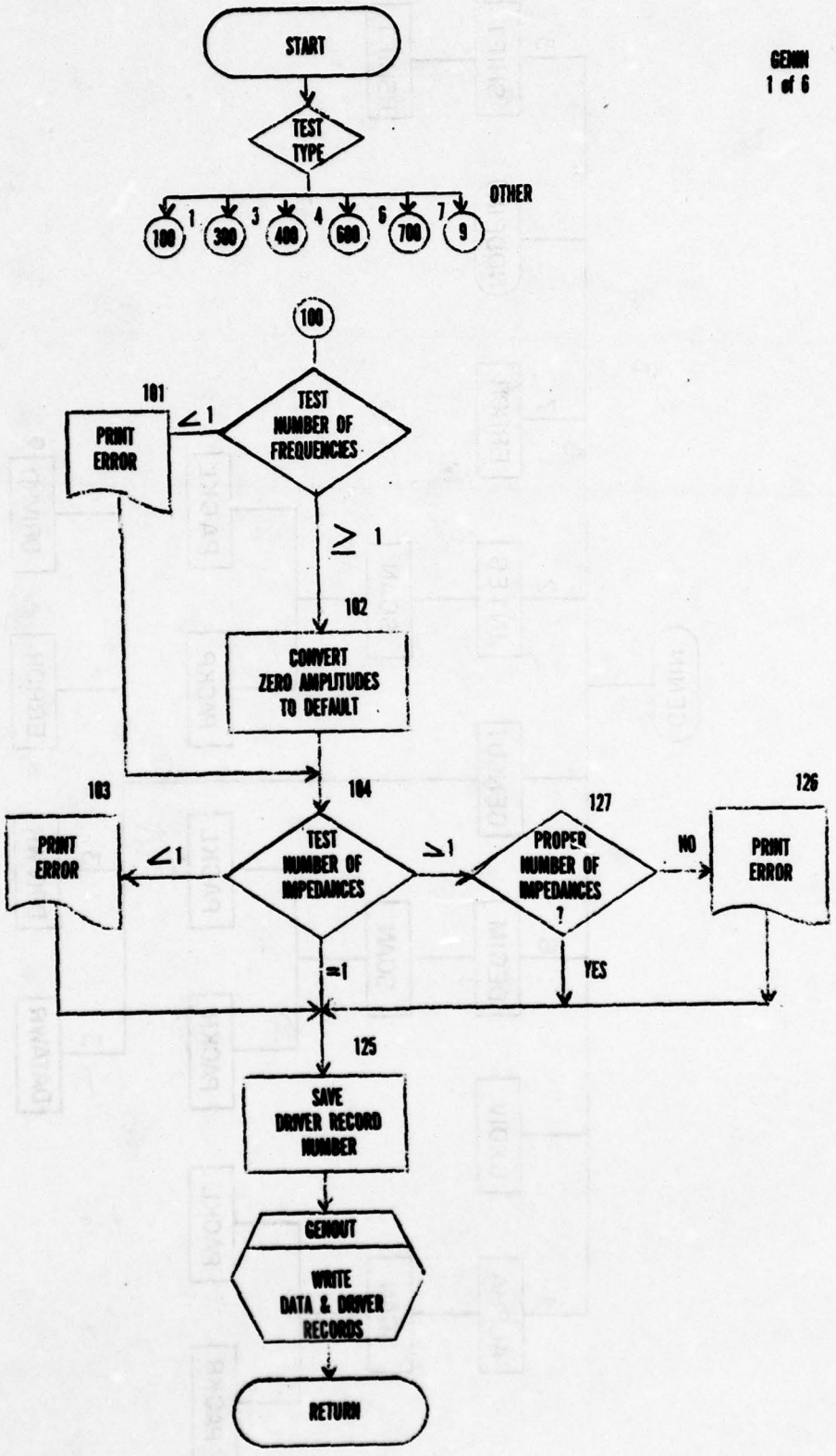
If errors were encountered in the input deck during Phase 0 processing (JABT = 1), NCAP execution terminates with a CALL EXIT.

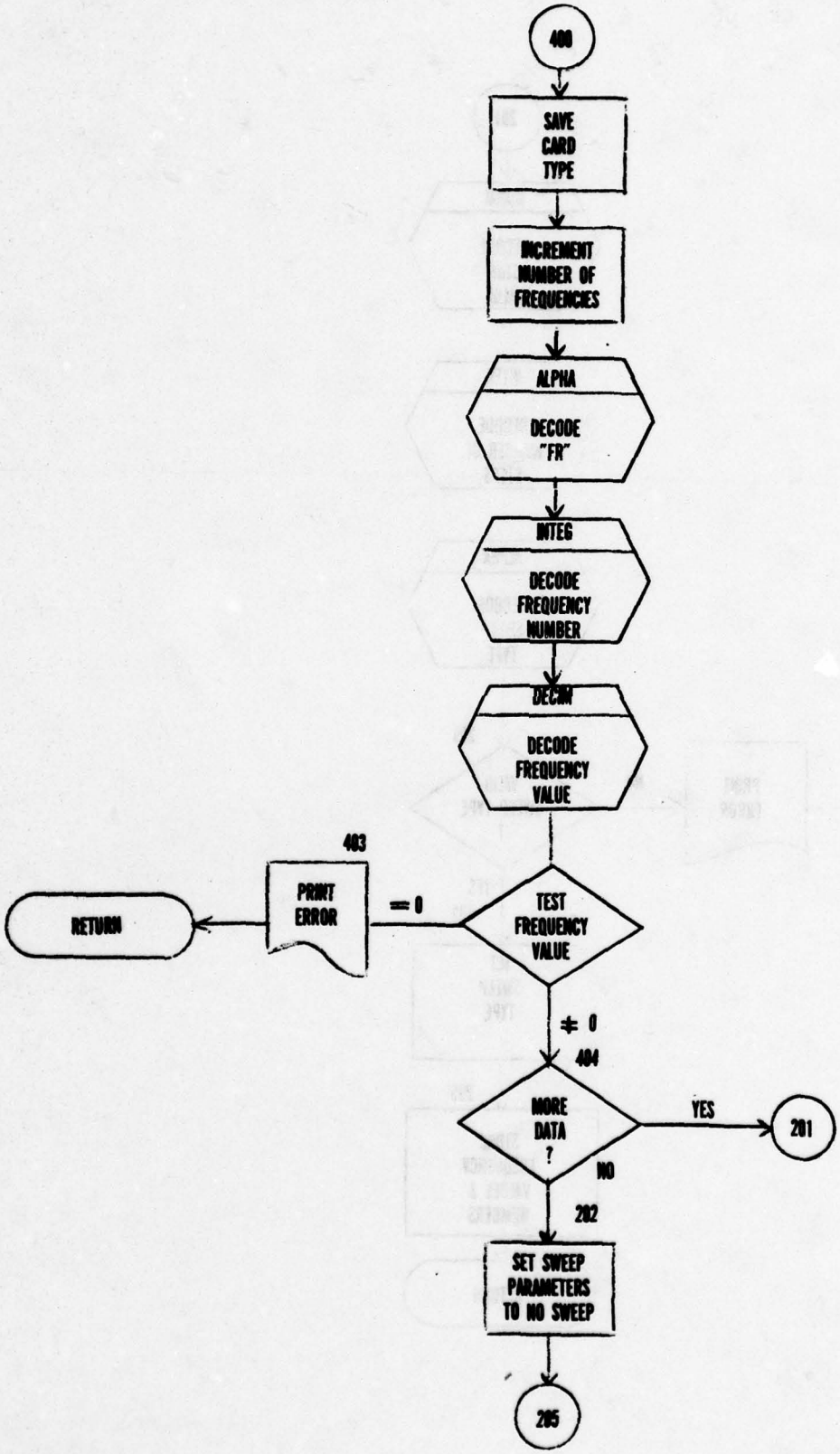
Otherwise the END driver record is written to file 21 by

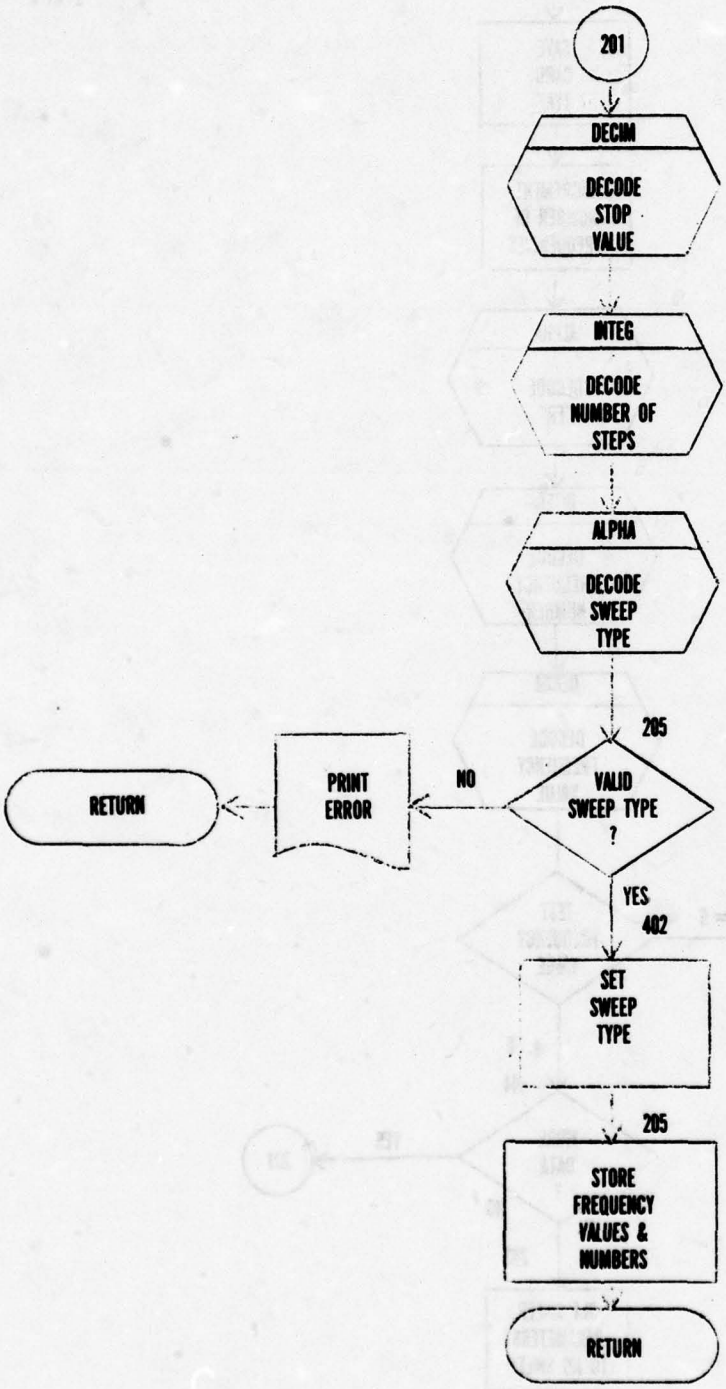
subroutine DRVOUT. The END driver record is identified by MODE=16. Because there is no data associated with the end function (it simply serves as a delimiter), the driver parameters STADD and LNGTH are not applicable. The MISC driver parameters are not used.

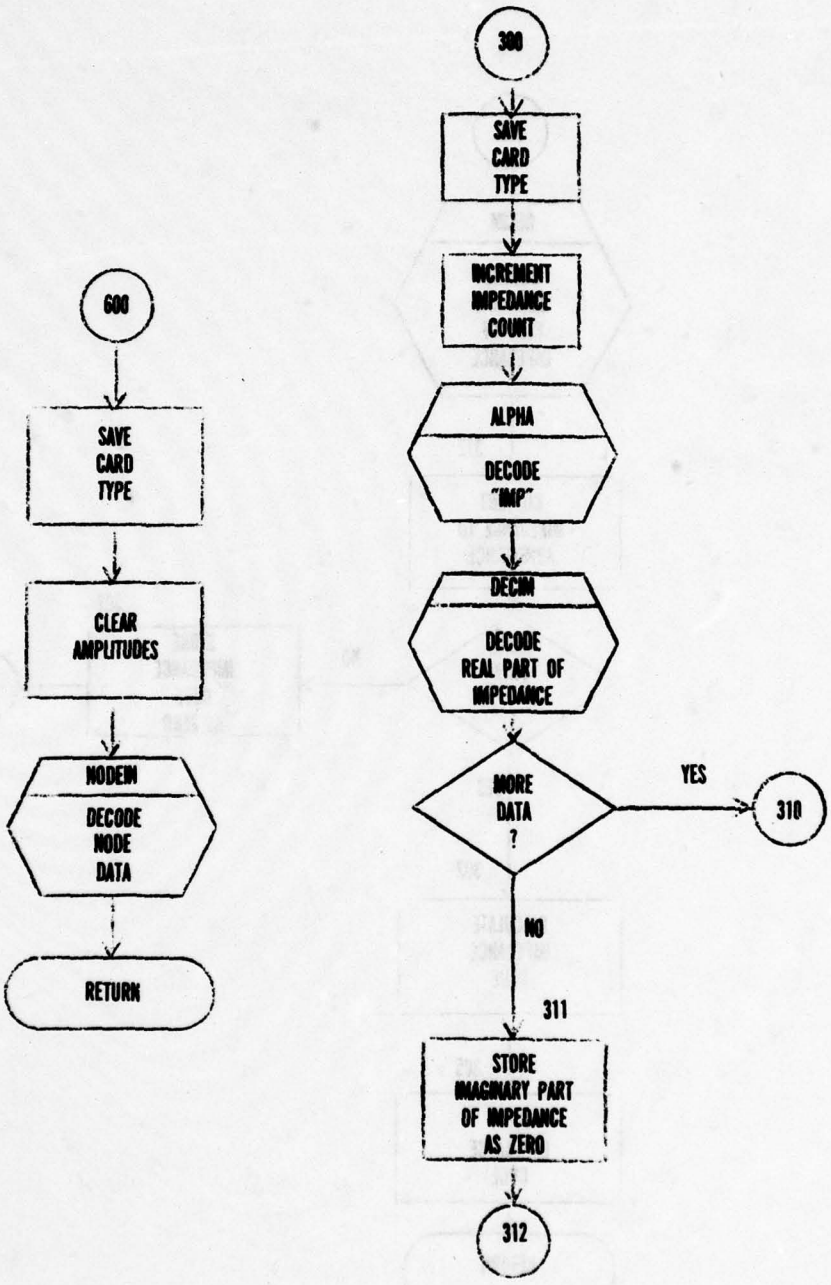
The Phase Ø processing is closed by creating the driver file header and writing it to the first record of file 21. The header record is identified by MODE=1 and contains the next available file 2Ø storage location in MISC(1) and MISC(4). The other driver parameters are not used. The header record is written to the driver file by subroutine DRVOUT and program control transfers to subroutine PHASE1.

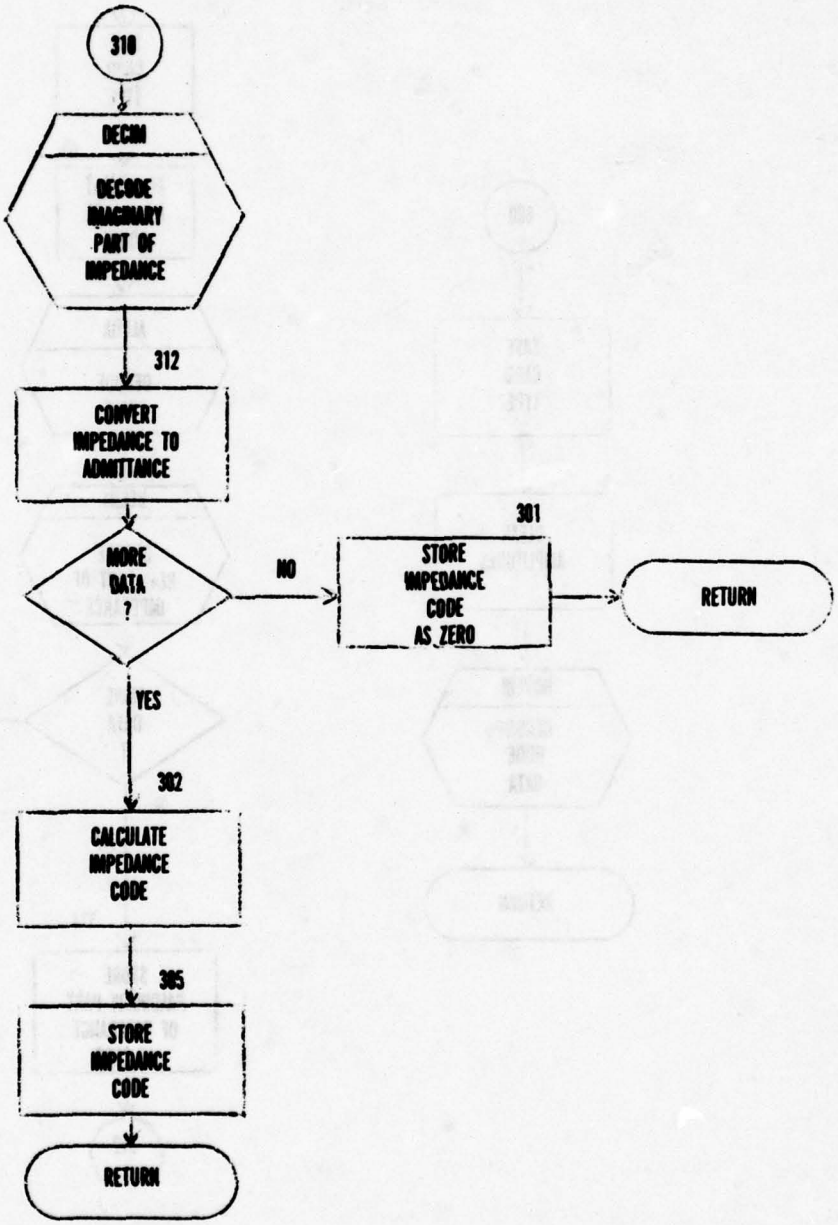

GENIN
4
ALPHA
CXDIV
6
DECIM
GENOUT
2
INTEG
7
ERROR
NODEIN
3
SHIFT
SCAN
SCAN
SCAN
RSHIFT
PACKR
PACKL
PACKR
PACKL
PACKR
PACKL
3
3
DATAWR
DRIVWR
ERROR
DRIVRD

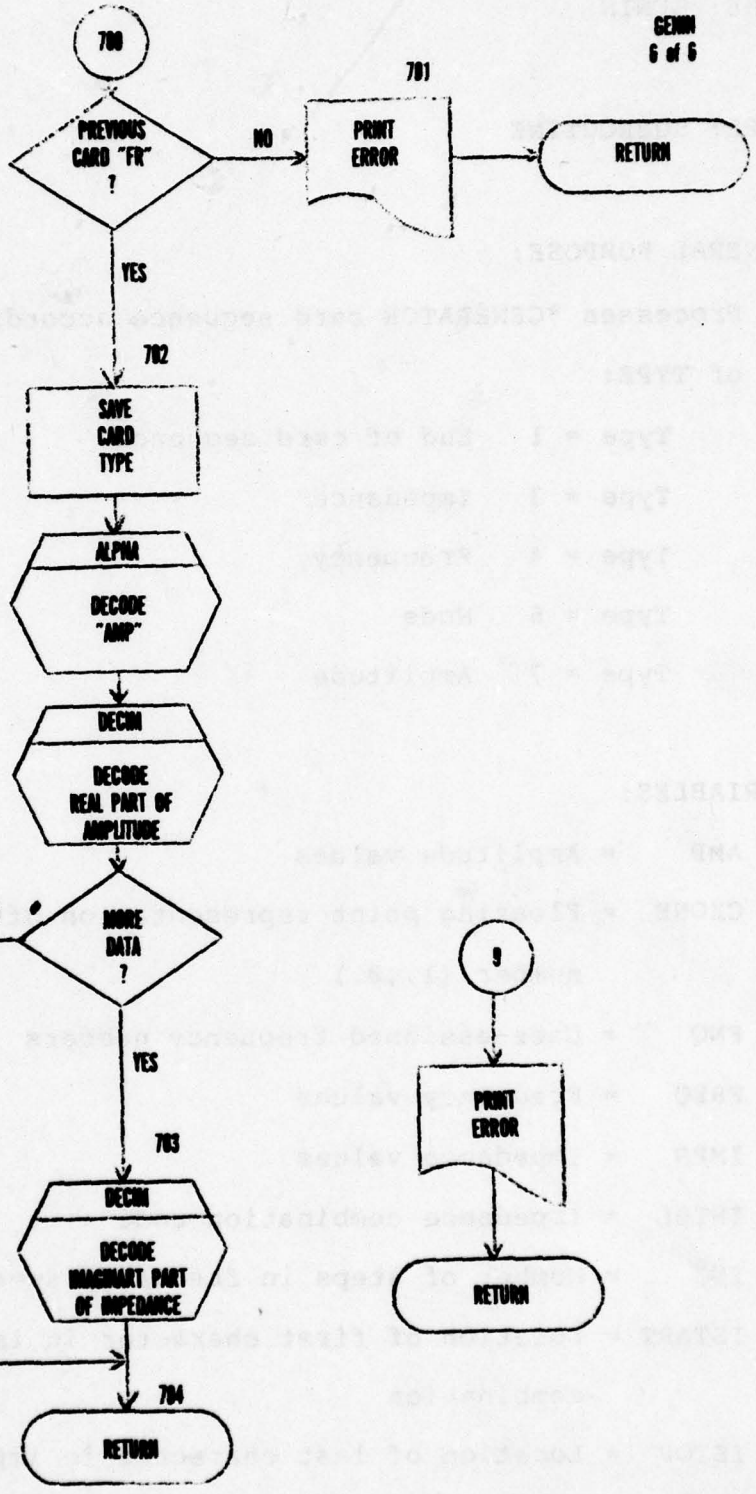

START
TEST TYPE
OTHER
1
3
4
6
7
100
300
400
600
700
9
100
101
PRINT ERROR
< 1
TEST NUMBER OF FREQUENCIES
≥ 1
102
CONVERT ZERO AMPLITUDES TO DEFAULT
103
PRINT ERROR
104
TEST NUMBER OF IMPEDANCES
< 1
≥ 1
127
PROPER NUMBER OF IMPEDANCES ?
NO
126
PRINT ERROR
=1
YES
125
SAVE DRIVER RECORD NUMBER
GENOUT
WRITE DATA & DRIVER RECORDS
RETURN


400
SAVE CARD TYPE
INCREMENT NUMBER OF FREQUENCIES
ALPHA
DECODE "FR"
INTEG
DECODE FREQUENCY NUMBER
DECIM
DECODE FREQUENCY VALUE
403
PRINT ERROR
= 0
RETURN
TEST FREQUENCY VALUE
≠ 0
404
MORE DATA ?
YES
201
NO
202
SET SWEEP PARAMETERS TO NO SWEEP
205


201
DECIM
DECODE STOP VALUE
INTEG
DECODE NUMBER OF STEPS
ALPHA
DECODE SWEEP TYPE
205
VALID SWEEP TYPE ?
NO
PRINT ERROR
RETURN
YES
402
SET SWEEP TYPE
205
STORE FREQUENCY VALUES & NUMBERS
RETURN


600
SAVE CARD TYPE
CLEAR AMPLITUDES
NODEIN
DECODE NODE DATA
RETURN
300
SAVE CARD TYPE
INCREMENT IMPEDANCE COUNT
ALPHA
DECODE "IMP"
DECIM
DECODE REAL PART OF IMPEDANCE
MORE DATA ?
YES
310
NO
311
STORE IMAGINARY PART OF IMPEDANCE AS ZERO
312


310
DECIM
DECODE IMAGINARY PART OF IMPEDANCE
312
CONVERT IMPEDANCE TO ADMITTANCE
MORE DATA ?
NO
301
STORE IMPEDANCE CODE AS ZERO
RETURN
YES
302
CALCULATE IMPEDANCE CODE
305
STORE IMPEDANCE CODE
RETURN


700
GENIN
6 of 6
701
PREVIOUS CARD "FR" ?
NO
PRINT ERROR
RETURN
YES
702
SAVE CARD TYPE
ALPHA
DECODE "AMP"
DECIN
DECODE REAL PART OF AMPLITUDE
NO
MORE DATA ?
YES
703
DECIN
DECODE IMAGINART PART OF IMPEDANCE
704
RETURN
9
PRINT ERROR
RETURN

NAME: GENIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *GENERATOR card sequence according to the value of TYPE:

Type = 1	End of card sequence
Type = 3	Impedance
Type = 4	Frequency
Type = 6	Node
Type = 7	Amplitude

VARIABLES:

AMP = Amplitude values

CXONE = Floating point representation of complex number (1.,0.)

FNO = User-assigned frequency numbers

FREQ = Frequency values

IMPS = Impedance values

IMTBL = Impedance combination code

INC = Number of steps in frequency sweep

ISTART = Location of first character in impedance combination

ISTOP = Location of last character in impedance combination + 1

ISW = End of data switch: Off = 0, On = 1

ITSV = Type code of last parameter card processed

ITYPE = Sweep type code: 0 = No sweep
1 = Linear
2 = Logarithmic

JRECO = Address of driver record

NFREQ = Number of frequencies this generator

NIMP = Number of impedances this generator

RBLANK = Right-justified alphanumeric blank

STRING = Alphanumeric string representation of input value

STOP = Stop value for sweep

TYPES = Valid frequency sweep types

SUBROUTINES CALLED:

ALPHA CXDIV DECIM ERROR

GENOUT INTEG NODEIN

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine GENIN is the input processor for GENERATOR card sequences. Its function is to decode the generator input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine GENIN causes a single NCAP input card to be processed. The card image and its type code are transmitted through common storage to subroutine GENIN which processes the card according to its type as follows:

TYPE=1 End of Card Sequence

The closing of a generator definition begins by testing the frequency and impedance specifications for errors. Each generator definition must contain at least one frequency card (NFREQ.GT.0) and either a constant impedance (NIMP=1) or one impedance card for each possible frequency combination (NIMP=$2^{NFREQ}-1$). If an improper number of frequencies and/or impedances have been specified, subroutine GENIN exits with an appropriate error condition.

Any undefined amplitude values are assigned the default value:

```
AMP (1,I) = 1.0
AMP (2,I) = 0.0
```

The amplitude data is then stored in the common data buffer at BUFF(11) - BUFF(30). Subroutine GENOUT is called to complete the definition of the generator driver and data records and to write them to disk, after which subroutine GENIN returns to the calling program.

TYPE=3 Impedance Card

The card type card is saved in ITSV, the number of impedances NIMP is incremented, and the IMP identifier is decoded by subroutine ALPHA. The real part of the impedance value is decoded and formatted by subroutine DECIM and stored in IMPS(1, NIMP). If the end-of-data is encountered (ISW=1), the imaginary part of the impedance value is assigned the default value zero. Otherwise the imaginary part of the impedance value is decoded and formatted by subroutine DECIM and stored in IMPS(2, NIMP). The impedance value is then converted to an admittance by subroutine CXDIV.

At this point the end-of-data switch ISW is used to determine the presence of an impedance combination on the card. If the end-of-data switch is on (ISW=1), subroutine GENIN exits after setting the impedance code BUFF(NIMP + 40) = 0 to indicate a constant impedance.

If the end-of-data is off (ISW=0), the impedance combination is extracted from the card image, converted to an impedance code, and appended to the impedance code table at BUFF(NIMP + 40). Each digit of the impedance combination is represented in the impedance code IMTBL by the bit position of like number. For example, if the digit 3 is included in the combination, the 3rd bit of IMTBL is turned on by IMTBL = 2^2. Beginning with the first digit of the combination stored at CARD (IPOS), each digit is mapped to the proper bit position and accumulated in IMTBL as follows:

IMTBL = IMTBL + (2** (CARD(J-1) -1))

This process continues under the control of the index J until either CARD(J) is a blank or six digits have been used. The resulting code is placed in the impedance code table at BUFF(NIMP + 40) and subroutine GENIN returns to the calling program.

TYPE=4 Frequency Card

The card type code is saved in ITSV, the number of frequencies NFREQ is incremented, and the FR identifier is decoded by subroutine ALPHA. The frequency number is decoded and formatted by subroutine INTEG and stored in FNO(NFREQ). The frequency value is decoded and formatted by subroutine DECIM and stored in FREQ(NFREQ). If the frequency number is zero, subroutine GENIN exits with an appropriate error condition.

At this point the end-of-data switch ISW is used to determine the presence of frequency sweep parameters on the card. If the end-of-data switch is on (ISW=1), the parameters ITYPE and INC are set to indicate no sweep. Otherwise the STOP, INC, and ITYPE parameters are decoded and formatted by subroutine DECIM, INTEG, and ALPHA respectively.

The sweep type is tested against a table of valid alphanumeric types. If no match is found, subroutine GENIN exits with an appropriate error condition. If a match is found, the sweep type is encoded in ITYPE(NFREQ), the frequency number and value are placed in the common data buffer BUFF, and subroutine

GENIN returns to the calling program.

TYPE=6 Node Card

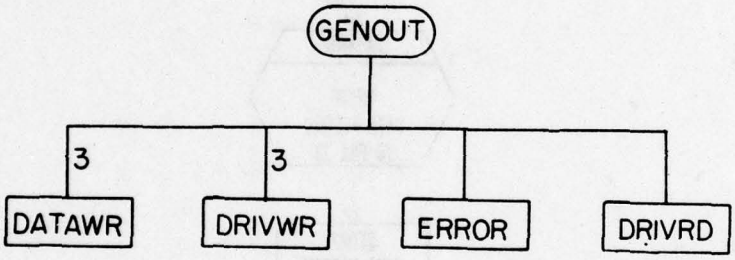
The card type code is saved in ITSV and the amplitude data area AMP is cleared. The nodes of connection are extracted from the card image, formatted, and stored in MISC(3) and MISC(4) of the driver record by subroutine NODEIN. Subroutine GENIN returns to the calling program.

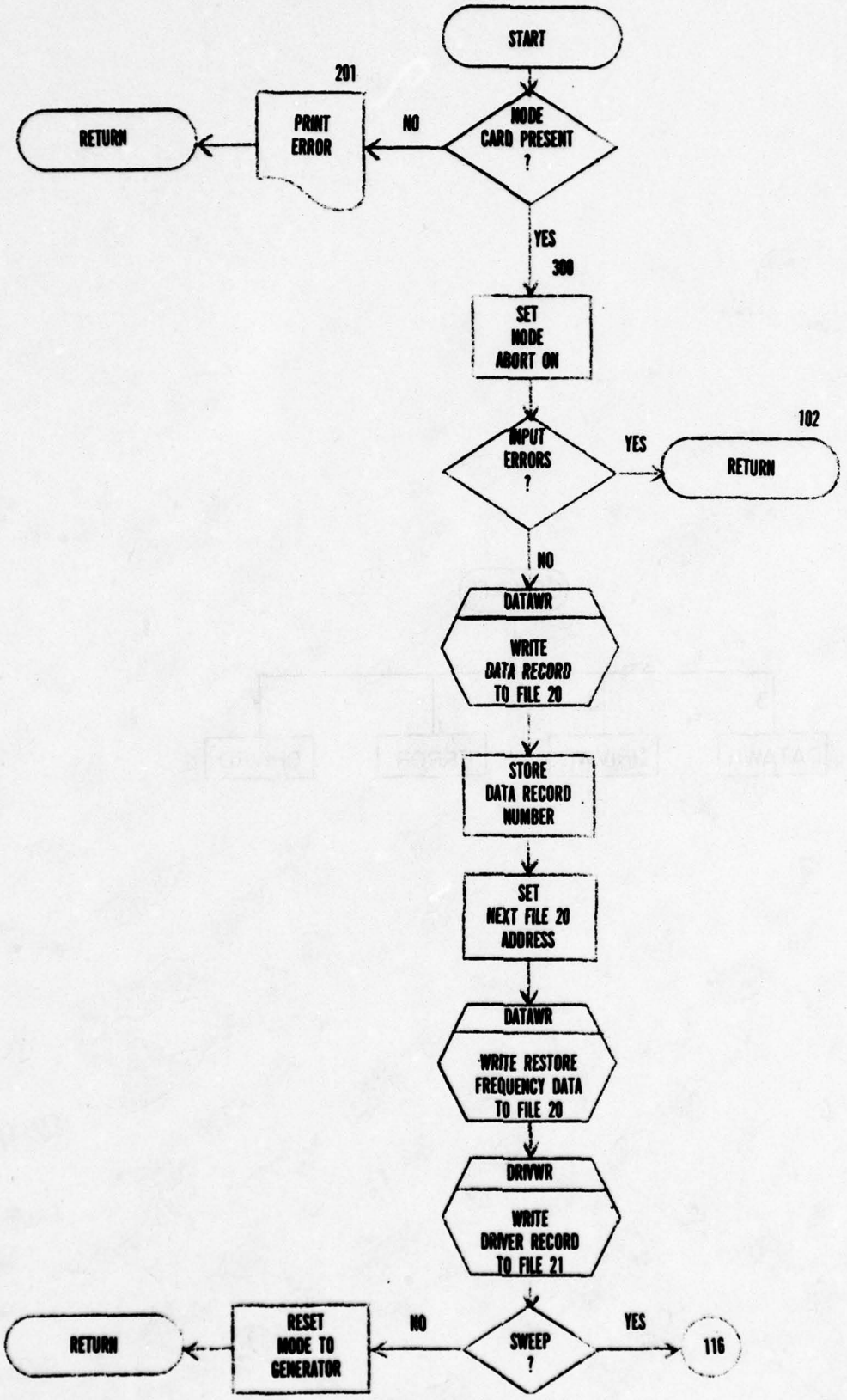
TYPE=7 Amplitude Card

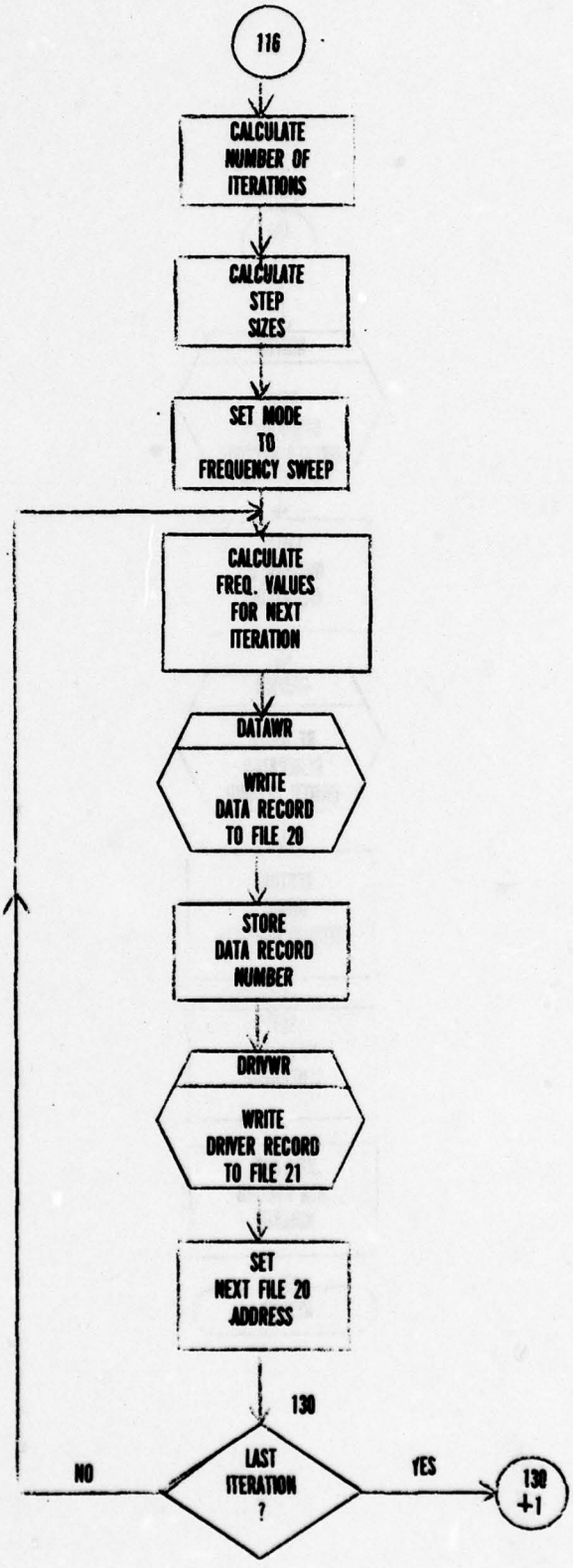
To assure that amplitude data are properly associated with frequency data, each amplitude card must physically follow a frequency card in the input deck. Therefore if the last card processed was not a frequency card (ITSV $\neq$ 4), subroutine GENIN exits with an appropriate error condition.

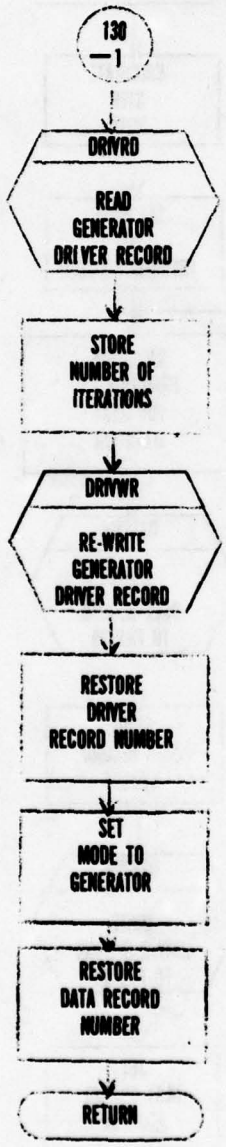
Otherwise the card type code is saved in ITSV and the AMP identifier is decoded by subroutine ALPHA. The real and imaginary parts of the complex amplitude are decoded and formatted by subroutine DECIM and stored in AMP(1, NFREQ) and AMP(2, NFREQ).

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


GENOUT
3
3
DATAWR
DRIVWR
ERROR
DRIVRD


START
201
PRINT ERROR
RETURN
NO
MODE CARD PRESENT ?
YES
300
SET MODE ABORT ON
INPUT ERRORS ?
YES
102
RETURN
NO
DATAWR
WRITE DATA RECORD TO FILE 20
STORE DATA RECORD NUMBER
SET NEXT FILE 20 ADDRESS
DATAWR
WRITE RESTORE FREQUENCY DATA TO FILE 20
DRIVWR
WRITE DRIVER RECORD TO FILE 21
RETURN
RESET MODE TO GENERATOR
NO
SWEEP ?
YES
116


116
CALCULATE NUMBER OF ITERATIONS
CALCULATE STEP SIZES
SET MODE TO FREQUENCY SWEEP
CALCULATE FREQ. VALUES FOR NEXT ITERATION
DATAWR
WRITE DATA RECORD TO FILE 20
STORE DATA RECORD NUMBER
DRIVWR
WRITE DRIVER RECORD TO FILE 21
SET NEXT FILE 20 ADDRESS
130
LAST ITERATION ?
NO
YES
130 +1


130
—1
DRIVRD
READ
GENERATOR
DRIVER RECORD
STORE
NUMBER OF
ITERATIONS
DRIVWR
RE-WRITE
GENERATOR
DRIVER RECORD
RESTORE
DRIVER
RECORD NUMBER
SET
MODE TO
GENERATOR
RESTORE
DATA RECORD
NUMBER
RETURN

NAME: GENOUT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Tests for errors in generator and modify generator input, writes input data to file 20 and driver record to file 21, creates and writes frequency sweep data and driver records

VARIABLES:

FNO = User-assigned frequency numbers

FREQ = Frequency values

INC = Number of steps in frequency sweep

ITYPE = Sweep type code; 0 = No sweep
1 = Linear
2 = Logarithmic

JRECO = Address of generator or modify driver record

NFREQ = Number of frequencies this generator

NIT = Maximum number of sweep iterations this generator

STRING = Alphanumeric string representation of input value

STEP = Sweep increment value

STOP = Stop value for sweep

SUBROUTINES CALLED·

DATAWR DRIVRD DRIVWR ERROR

CALLING PROGRAMS·

GENIN MODGIN

DESCRIPTION·

Subroutine GENOUT handles the output of data and driver records for the generator input processing subroutines GENIN and MODGIN, closing the generator and modify generator definitions.

First the input processed by GENIN or MODGIN is tested for errors. If the node card is missing (NABT≠0) or the input error switch is on (JABT=1), return is made to the calling program with an appropriate error condition. Otherwise the data record is written to file 20 from the common data buffer BUFF by subroutine DATAWR.

The generator data record contains 230 words allocated as follows·

BUFF(1)-(10)	Frequency Values
BUFF(11)-(30)	Complex Amplitudes
BUFF(31)-(40)	User-assigned Frequency Numbers
BUFF(41)-(103)	Impedance Combination Codes
BUFF(104)-(230)	Complex Impedance Values

After the data record has been written to disk, its record number is stored at STADD and the next available file 20 storage location is updated to point beyond the generator data by DATREC=DATREC+230. The frequency values, amplitudes, and

frequency numbers stored in BUFF(1)-(40) are then rewritten to the data file at record number DATREC and the next available file 20 storage location is updated to point beyond the re-written data by DATREC=DATREC+40. (This re-write file is later used by Phase 4 to restore the original frequency data after execution of a frequency sweep.)

The driver record parameters defined in the input processor, GENIN or MODGIN, are transmitted through common storage to subroutine GENOUT. Here the remaining driver parameters are defined and the record is written by subroutine DRIVWR to file 21 from the first ten words of global common. The complete driver record is comprised of:

MODE=3 or 21	Generator or Modify Generator Identifier
STADD	Data File Record Number
LNGTH=230	Length of Data Record in Words
MISC(1)=NFREQ	Number of Frequencies
MISC(2)=NIMP	Number of Impedances
MISC(3)	Positive Node of Connection
MISC(4)	Negative Node of Connection
MISC(6)=NIT	Number of Sweep Iterations

Other MISC driver parameters are not used.

Unlike most NCAP input card sequences which map to a single driver and data record, generator and modify generator card sequences result in one or more driver records, each with an associated data record. The number of driver records is related to the number of sweep iterations defined in the card sequence.

If frequency sweeping is not specified in the card sequence (i.e., ITYPE(I)=0 for all I=1, NFREQ), the card sequence has only a single MODE=3 or 21 driver record associated with it and subroutine GENOUT returns to the calling program without further processing.

Otherwise the sweep specifications must be translated into appropriate driver and data records. First the number of sweep iterations, NIT, is derived from the largest INC parameter specified in the generator or modify generator card sequence. (Each of the NIT sweep iterations causes one driver and data record to be created internally and appended to the disk files.)

Then a sweep increment STEP(I) is calculated for each frequency value in the generator definition. The increment is a function of the original frequency value FREQ(I) and the STOP, INC, and ITYPE parameters associated with that frequency as follows:

1) For ITYPE(I)=1, linear sweep is indicated and:

$$STEP(I) = \frac{STOP(I) - FREQ(I)}{INC(I) - 1.}$$

2) For ITYPE(I)=2, logarithmic sweep is indicated and·

$$STEP(I) = \left[\frac{STOP(I)}{FREQ(I)}\right]^{\frac{1.}{INC(I)-1.}}$$

After the increments have been calculated, the sweep driver and data records are created and appended to the disk files. Like standard driver records, sweep driver records are developed in the first ten words of global common. A frequency sweep driver record consists of:

MODE=12	Frequency Sweep Identifier
STADD	Data File Record Number
LNGTH=40	Length of Data Record in Words
MISC(1)=NFREQ	Number of Frequencies
MISC(2)=NIMP	Number of Impedances
MISC(3)	Positive Node of Connection
MISC(4)	Negative Node of Connection
MISC(6)=NIT	Number of Sweep Iterations

Other MISC driver parameters are not used.

Frequency sweep data records have the same structure as the first 40 words of the original generator data record and are developed in the common data buffer BUFF by an iterative process. The first sweep data record is derived from the original generator data by performing an incrementation operation on each frequency value BUFF(J), J=1,NFREQ according to its corresponding STEP(J), STOP(J), and ITYPE(J) parameters:

1) For ITYPE(J)=0, no sweep is indicated. No incrementation is performed and the frequency value remains

constant at BUFF(J).

2) For ITYPE(J)=1, linear sweep is indicated and the incrementation is additive:

BUFF(J)=BUFF(J) + STEP(J)

unless STOP(J) is exceeded, in which case the frequency remains constant at its last value.

3) For ITYPE(J)=2, logarithmic sweep is indicated and the incrementation is multiplicative:

BUFF(J)=BUFF(J) * STEP(J)

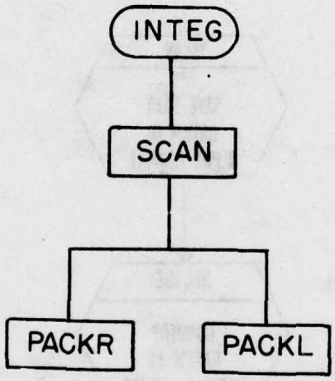
If STOP(J) is exceeded, the frequency value remains constant at STOP(J).

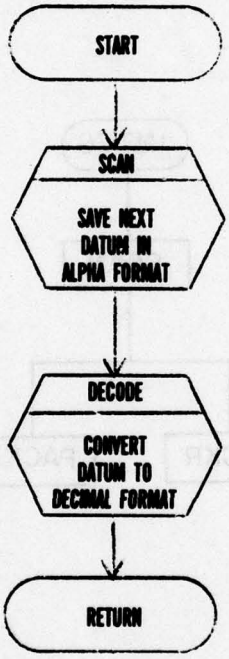
After every frequency value in the buffer has been updated, the new data record is written to file 20 at record number DATREC by subroutine DATAWR. The next available file 20 storage location is updated to DATREC=DATREC + 40 and subroutine DRIVWR is called to write the sweep driver record to file 21. The data record number STADD is updated and program control transfers back to process the next sweep iteration. The creation of sweep records continues under the control of the index I until all NIT sweep iterations have been translated into data and driver records.

The original generator or modify generator driver record is read from the driver file by subroutine DRIVRD according to the record number JRECO transmitted to subroutine GENOUT as an

argument. After storing the sweep iteration count NIT in MISC(6), the updated driver record is re-written by subroutine DRIVWR.

After the MODE is restored to 3 and the data file record number STADD is updated, subroutine GENOUT returns to the calling program.


INTEG
SCAN
PACKR
PACKL


START
SCAN
SAVE NEXT DATUM IN ALPHA FORMAT
DECODE
CONVERT DATUM TO DECIMAL FORMAT
RETURN

NAME: INTEG

TYPE: SUBROUTINE

GENERAL PURPOSE:

Decodes and formats an integer input value by converting the characters stored in CARD(IPOS) through CARD(J-1) to a fixed point number returned in K

VARIABLES:

CARD = Right-justified alphanumeric card image

IPOS = Position of first character to be formatted

ISW = End of data switch; off = 0, on = 1

J = Position of last character to be formatted+1

K = Integer value returned

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

SCAN

CALLING PROGRAMS:

GENIN LCIN LDSIN MODCIN NCIN

NDSIN NODEIN PLOTIN PSIN

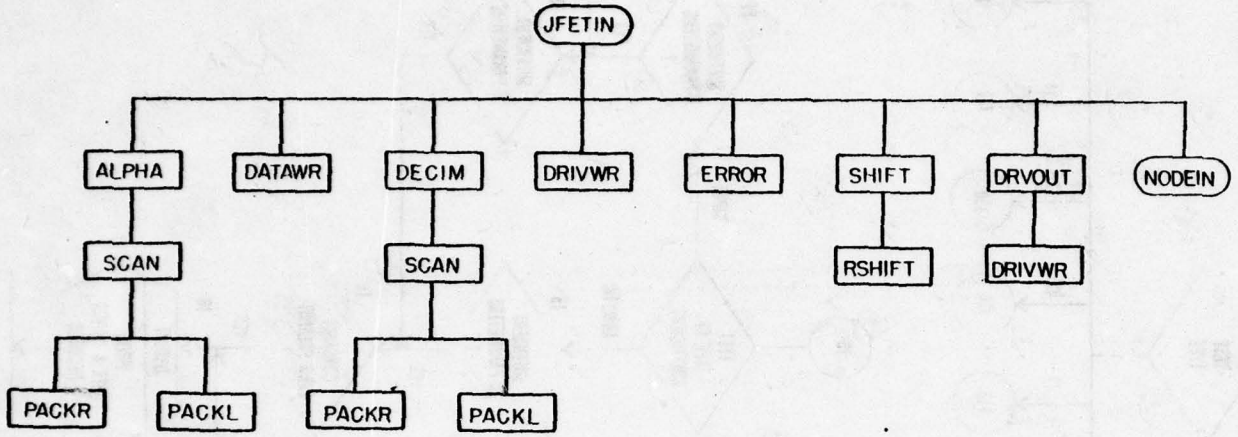
DESCRIPTION:

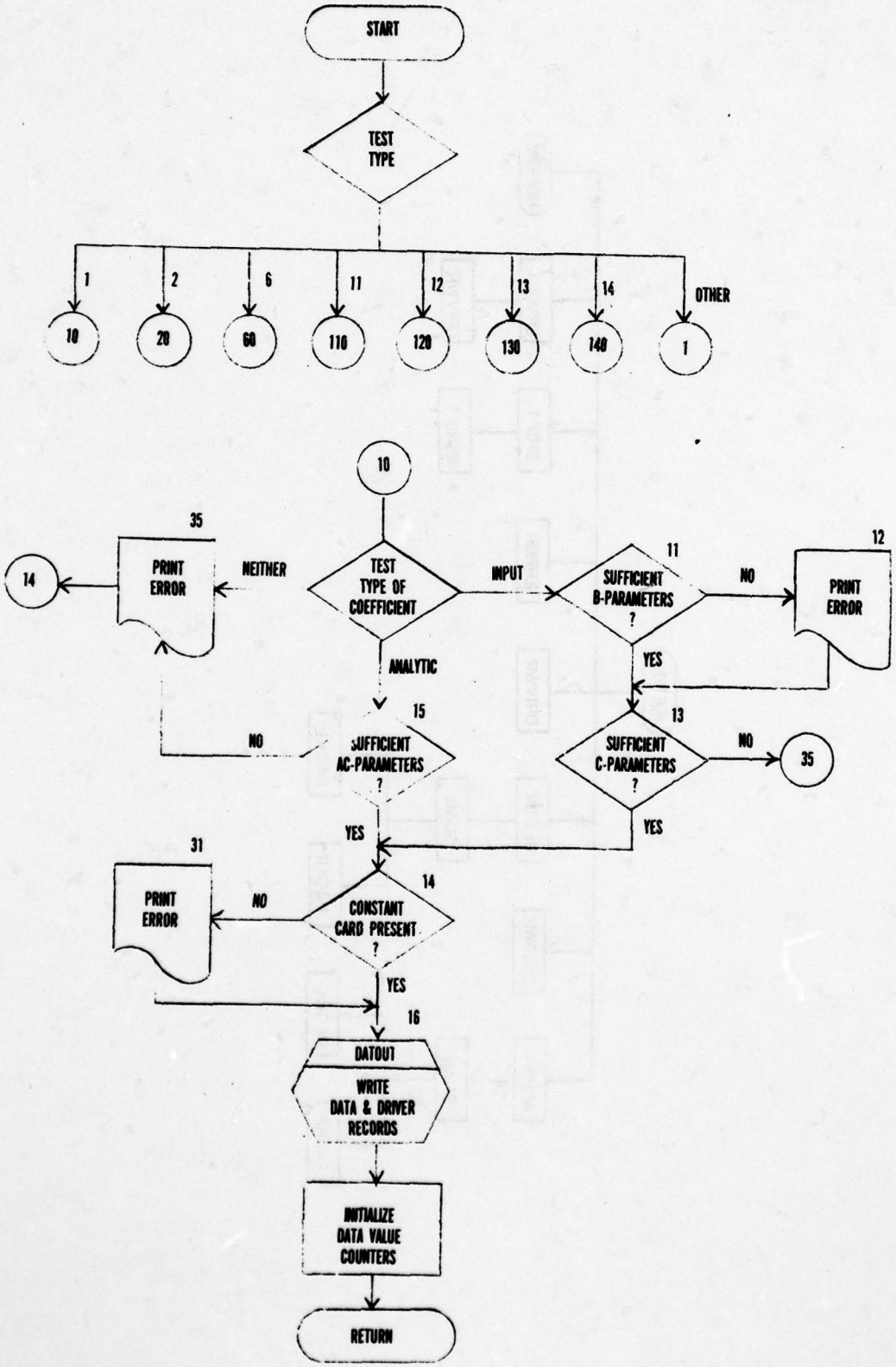
Subroutine INTEG decodes and formats a free-form integer input value. Input to the routine is the right-justified alphanumeric card image stored in the integer array CARD and the character position IPOS at which the datum begins.

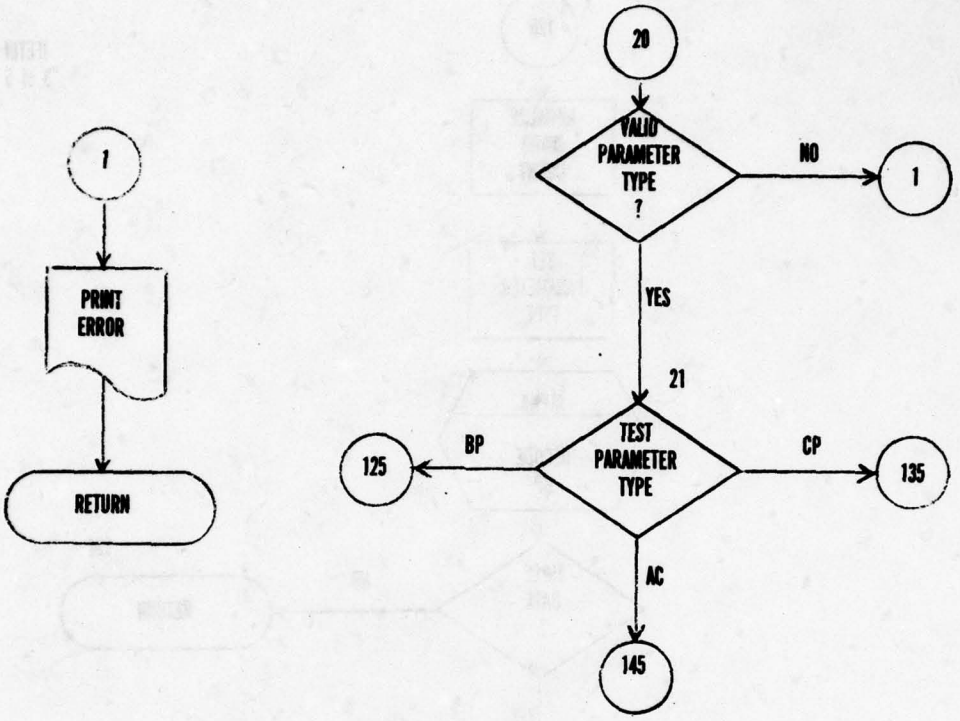
The contiguous characters CARD(IPOS) through CARD(J-1) which comprise the integer input value are extracted from the card image and converted to a right-justified alphanumeric string by subroutine SCAN. The character string returned in the integer array STRING is then converted from alphanumeric representation to an integer value K by:

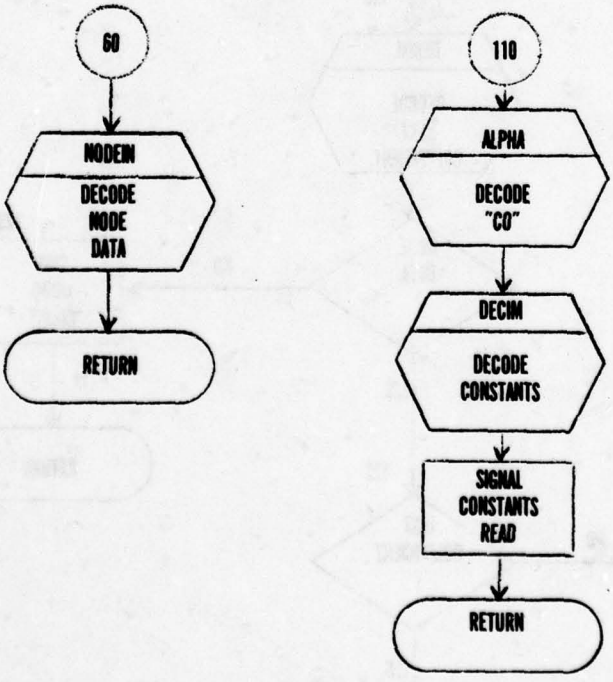
```
      DECODE (STRING,2) K
    2 FORMAT (I12)
```

Arguments returned by subroutine INTEG are the integer input value K, the updated character position IPOS which indicates the first character position of the next data value on the card, and the end-of-data switch ISW.


JFETIN
ALPHA
DATAWR
DECIM
DRIVWR
ERROR
SHIFT
DRVOUT
NODEIN
SCAN
SCAN
RSHIFT
DRIVWR
PACKR
PACKL
PACKR
PACKL

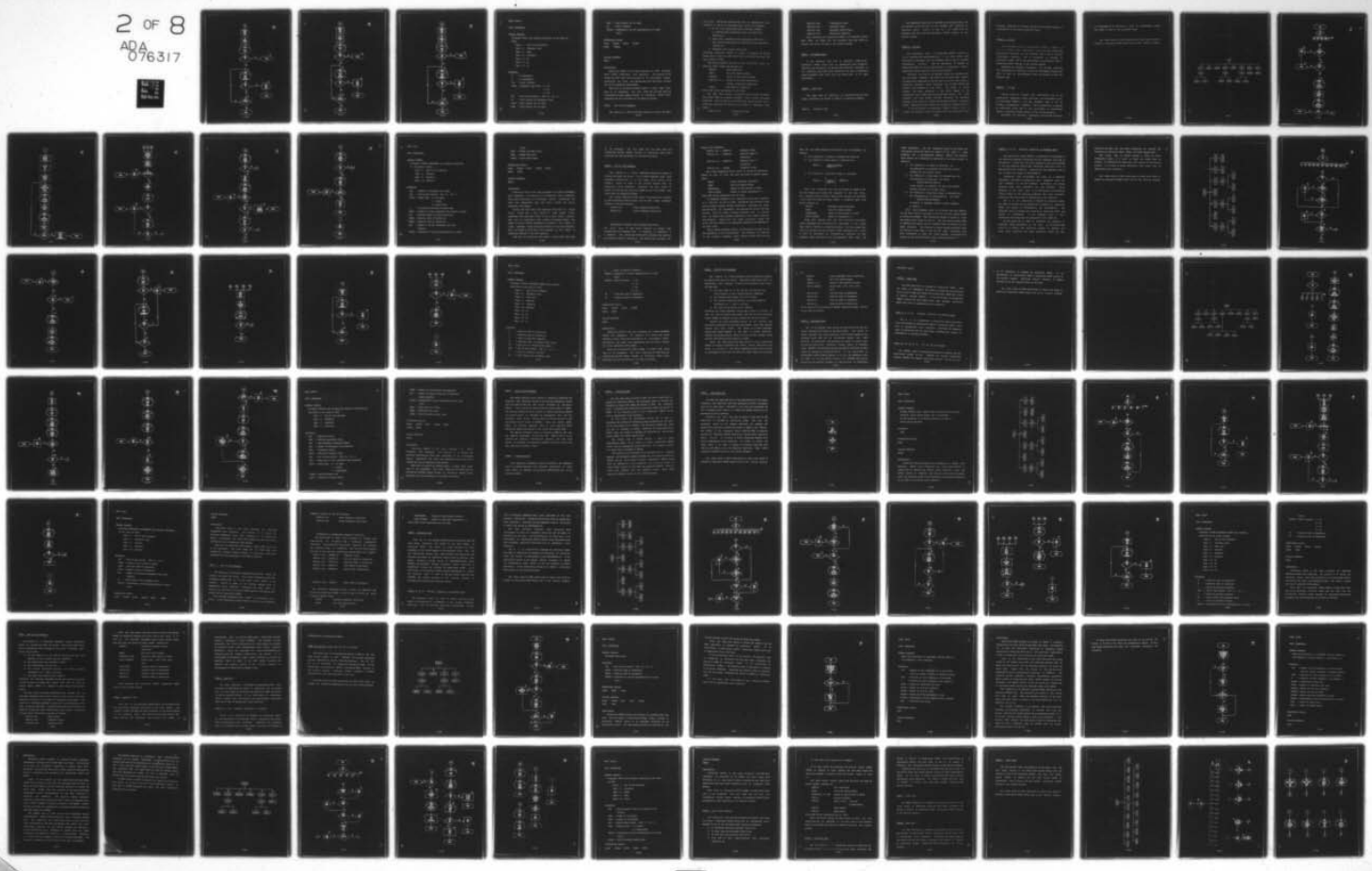

START
TEST TYPE
1
2
6
11
12
13
14
OTHER
10
20
60
110
120
130
140
1
10
TEST TYPE OF COEFFICIENT
NEITHER
35
PRINT ERROR
14
INPUT
11
SUFFICIENT B-PARAMETERS ?
NO
12
PRINT ERROR
YES
ANALYTIC
NO
15
SUFFICIENT AC-PARAMETERS ?
13
SUFFICIENT C-PARAMETERS ?
NO
35
YES
YES
31
PRINT ERROR
NO
14
CONSTANT CARD PRESENT ?
YES
16
DATOUT
WRITE DATA & DRIVER RECORDS
INITIALIZE DATA VALUE COUNTERS
RETURN

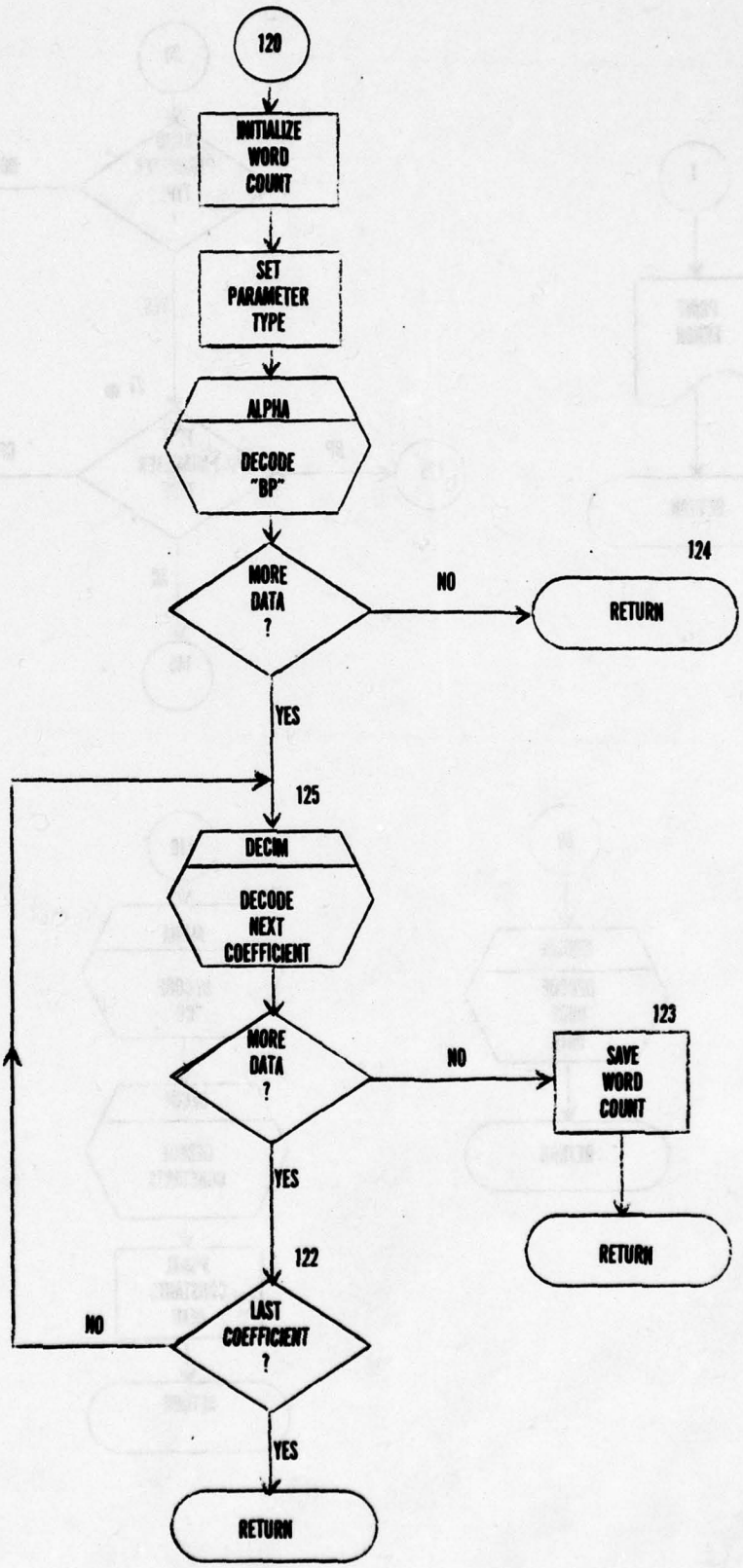

20
VALID PARAMETER TYPE ?
NO
1
YES
21
TEST PARAMETER TYPE
BP
125
CP
135
AC
145
1
PRINT ERROR
RETURN

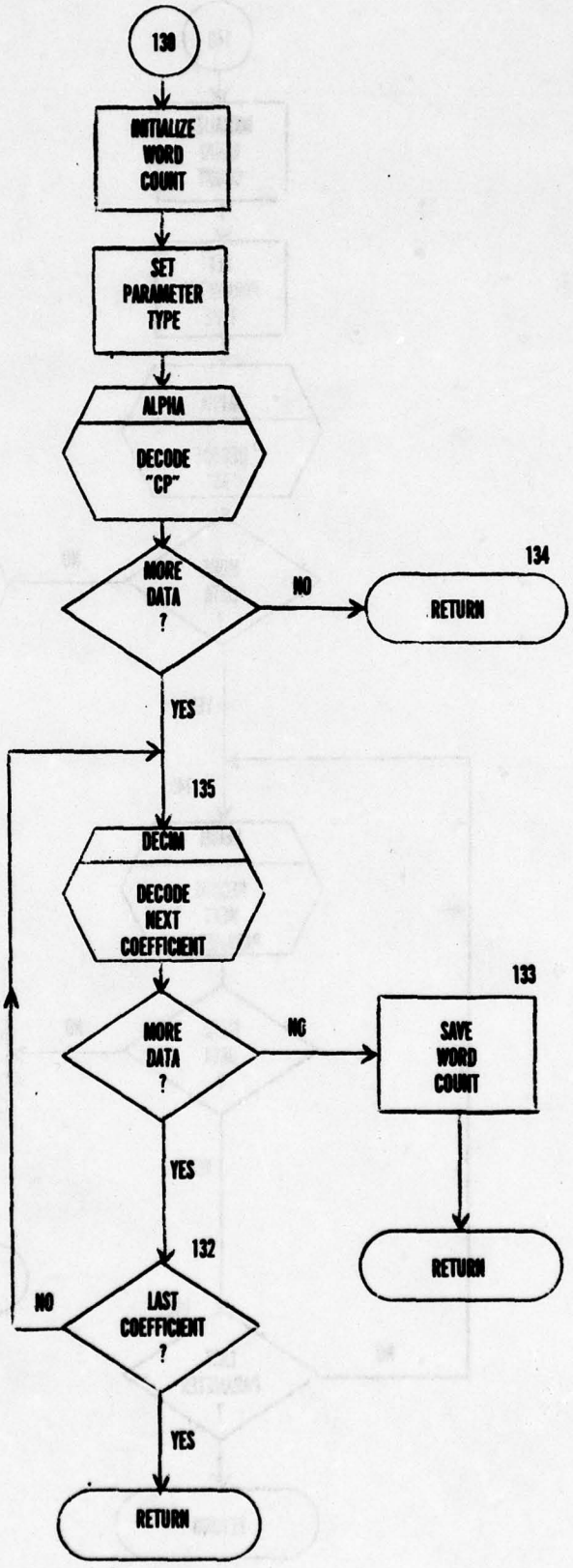

60
NODEIN
DECODE NODE DATA
RETURN
110
ALPHA
DECODE "CO"
DECIM
DECODE CONSTANTS
SIGNAL CONSTANTS READ
RETURN

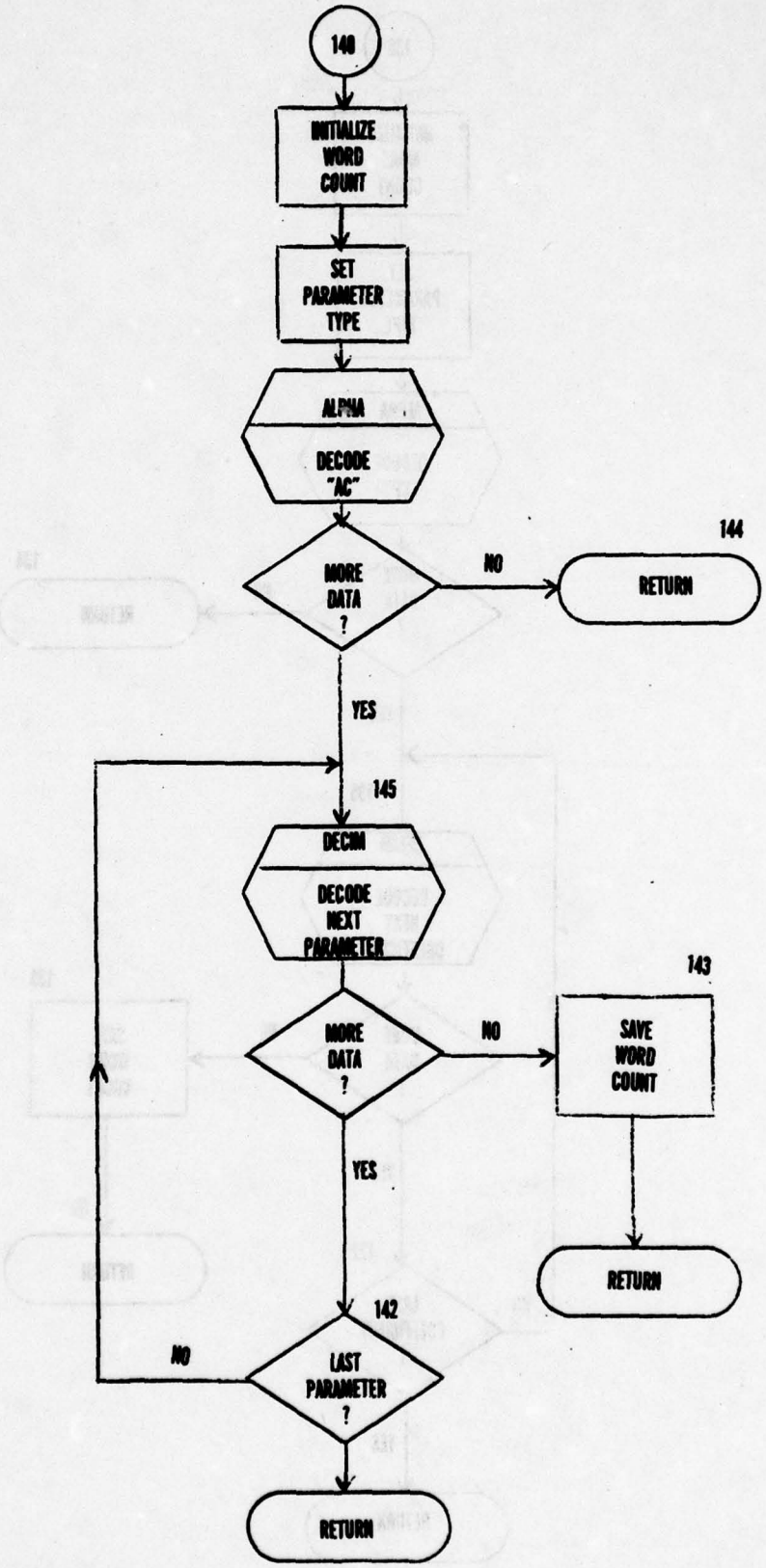
AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

2 OF 8
ADA 076317




120
INITIALIZE WORD COUNT
SET PARAMETER TYPE
ALPHA
DECODE "BP"
MORE DATA ?
NO
124
RETURN
YES
125
DECIM
DECODE NEXT COEFFICIENT
MORE DATA ?
NO
123
SAVE WORD COUNT
RETURN
YES
122
LAST COEFFICIENT ?
NO
YES
RETURN


130
INITIALIZE WORD COUNT
SET PARAMETER TYPE
ALPHA
DECODE "CP"
MORE DATA ?
NO
134
RETURN
YES
135
DECOM
DECODE NEXT COEFFICIENT
MORE DATA ?
NO
133
SAVE WORD COUNT
YES
RETURN
132
NO
LAST COEFFICIENT ?
YES
RETURN


140
INITIALIZE WORD COUNT
SET PARAMETER TYPE
ALPHA
DECODE "AC"
144
MORE DATA ?
NO
RETURN
YES
145
DECIM
DECODE NEXT PARAMETER
143
MORE DATA ?
NO
SAVE WORD COUNT
YES
RETURN
142
NO
LAST PARAMETER ?
RETURN

NAME: JFETIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *JFET card sequence according to the value of TYPE:

Type = 1 End of card sequence

Type = 2 Parameter card

Type = 6 Node

Type = 11 Constants

Type = 12 BP

Type = 13 CP

Type = 14 AC

VARIABLES:

B = B parameters

C = C parameters

CGD = Input constant

ICOEF = Parameter type code: 1 = BP
2 = CP
3 = AC

ISW = End of data switch: Off = 0, On = 1

NW = Word counter for parameter input

NXTAC = Word counter for AC input

NXTB = Word counter for BP input

NXTC = Word counter for CP input

RS = Input constant

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

ALPHA DATOUT DECIM DRIVWR

ERROR NODEIN

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine JFETIN is the input processor for JFET (Junction Field Effect Transistor) card sequences. Its function is to decode the JFET input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine JFETIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine JFETIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

The closing of a JFET definition begins by testing the input

for errors. Subroutine JFETIN exits with an appropriate error condition if any of the following input errors are detected·

1) No BP or CP coefficients were processed (NXTB=Ø) or (NXTC=Ø) when coefficient input was specified (MISC(2)=1)
2) Fewer than 8 parameters were processed (NXTAC.LT.8) when analytic generation of coefficients was specified (MISC(2)=2)
3) CONSTANTS card missing (MISC(1)=Ø)

Otherwise subroutine DATOUT is called to complete the driver record and to close the JFET definition by writing the driver and data records to disk.

The driver record is written to file 21 from the first ten words of global common and consists of:

MODE=2Ø	JFET Identifier
STADD	Data File Record Number
LNGTH=64	Length of Data File in Words
MISC(2)=1 or 2	Method of Deriving Coefficients· 1=Input, 2=Analytic generation
MISC(3)	Base Node of Connection

Other MISC driver parameters are not used.

The JFET data record, written to file 2Ø from the common data buffer BUFF is 64 words in length. This record allocates sufficient disk storage for the JFET input values as well as all other data generated internally by its model in subsequent NCAP phases:

BUFF(1)-(1Ø)	B-Parameters Input

BUFF(11)-(20)	C-Parameters Input
BUFF(21)-(22)	Constants Input
BUFF(23)-(32)	Nonlinear Coefficients
BUFF(33)-(64)	Admittance Submatrix

After returning from subroutine DATOUT, the parameter counts NXTB, NXTC, and NXTAC and the parameter type code ICOEF are cleared, and return is made to the calling program.

TYPE=2 Parameter Card

If the parameter type code is undefined (ICOEF.LT.0), subroutine JFETIN exits with an appropriate error condition. Otherwise the parameters are decoded and formatted according to the parameter code ICOEF which identifies the type of parameters being processed (1=BP, 2=CP, 3=AC) and causes them to be input accordingly.

TYPE=6 Node Card

The base node of connection is extracted from the card image, formatted, and stored in MISC(3) by subroutine NODEIN.

TYPE=11 Constants Card

The CONSTANTS identifier is decoded by subroutine ALPHA, and the constant values CGD and RS are decoded and formatted by subroutine DECIM. MISC(1) is set to 1 to signal that the constants have been read and subroutine JFETIN returns to the calling program.

TYPE=12 BP Card

The B-parameter count is initialized (NXTB=1), MISC(2) is set to indicate that coefficients are to be read rather than analytically generated, and the parameter code is set to indicate B-parameters (ICOEF=1). The BP identifier is decoded by subroutine ALPHA and if the end-of-data is encountered (ISW=1), subroutine JFETIN returns to the calling program.

Otherwise from one to ten decimal values are extracted from the card image, formatted, and stored in the common data buffer. B-coefficients appearing on the BP card as well as those input on unlabelled parameter cards following the BP card (TYPE=2 and ICOEF=1) are processed at this point. The number of values decoded and their placement in the buffer depend on the B-parameter count, NXTB, and the end-of-data switch, ISW. NXTB contains the next buffer location available for parameter storage, which begins at B(NXTB) and proceeds through successive buffer locations as data values are decoded by subroutine DECIM.

If an end-of-data is encountered (ISW=1) before B(10) is filled, the parameter count is reset to the next available buffer

location (NXTP=NW) and control returns to the calling program in anticipation of additional parameter cards.

TYPE=13 CP Card

The C-parameter count is initialized (NXTC=1). MISC(2) is set to indicate that coefficients are to be read rather than analytically generated, and the parameter code is set to indicate C-parameters (ICOEF=2). The CP identifier is decoded by subroutine ALPHA and if the end-of-data is encountered (ISW=1), subroutine JFETIN returns to the calling program.

Otherwise the decoding, formatting, and storage technique for C-parameters is identical to that of B-parameters except that NXTC is used for the parameter count and storage is from C(1) through C(10).

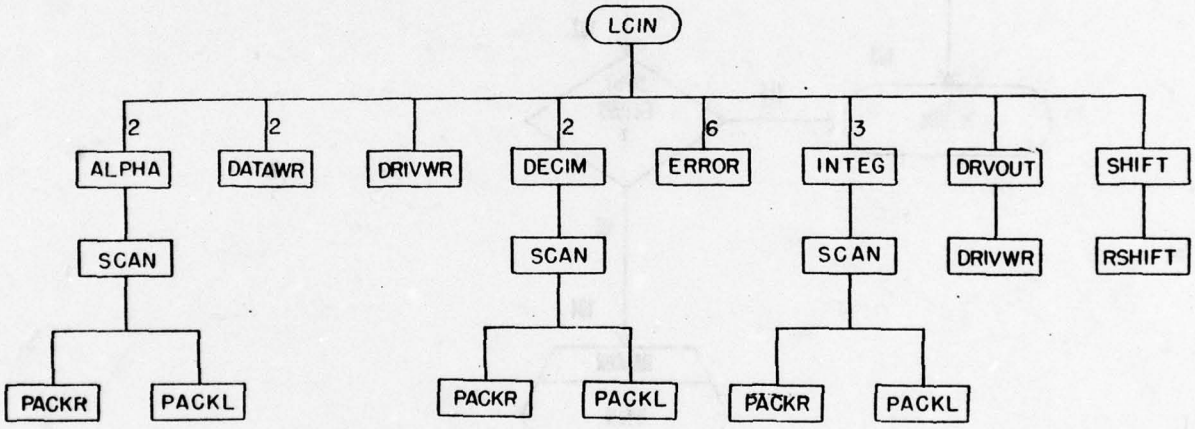
TYPE=14 AC Card

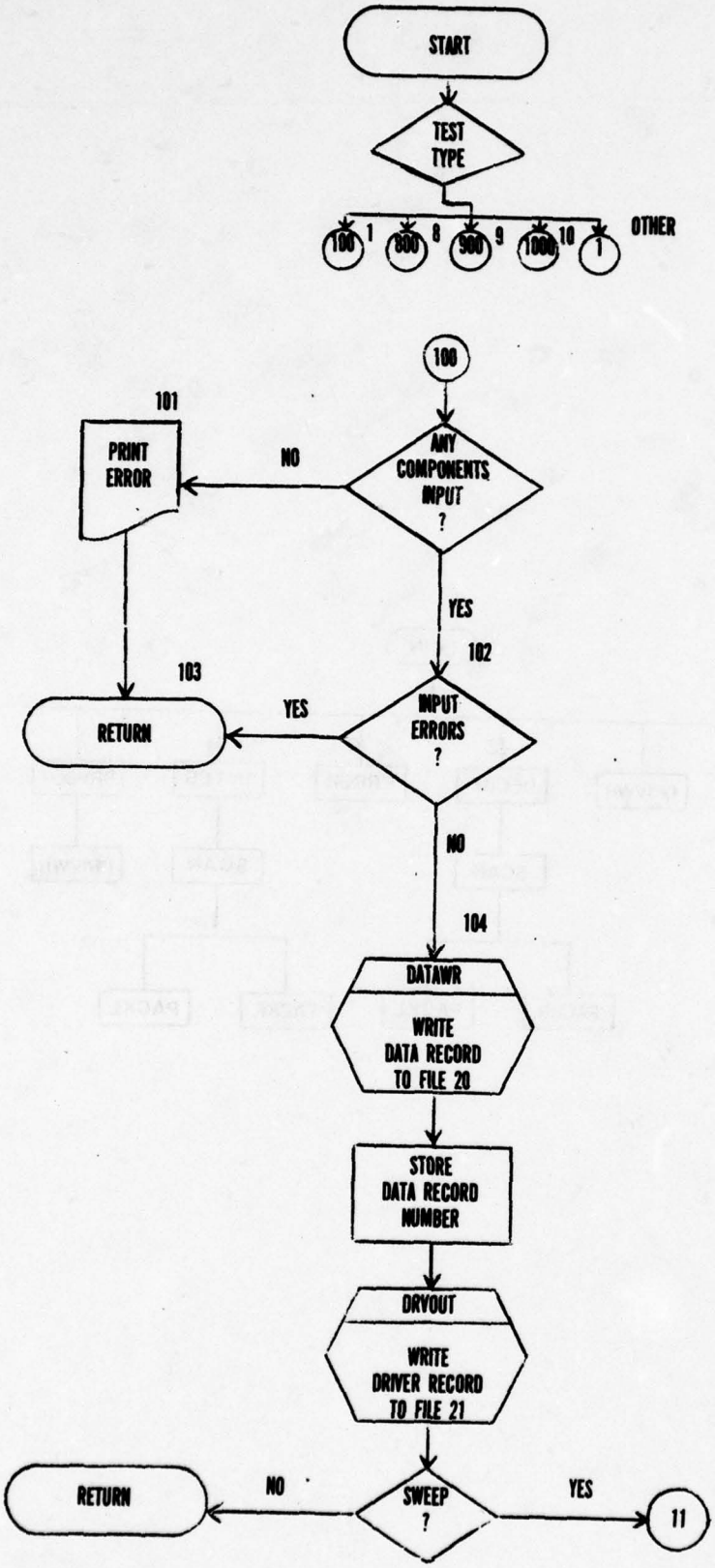
MISC(2) is set to indicate that coefficients are to be analytically generated rather than read, the AC-parameter count is initialized (NXTAC=1), and the parameter code is set to indicate AC-parameters (ICOEF=3). The AC identifier is decoded by subroutine ALPHA and if the end-of-data is encountered (ISW=1), subroutine JFETIN returns to the calling program.

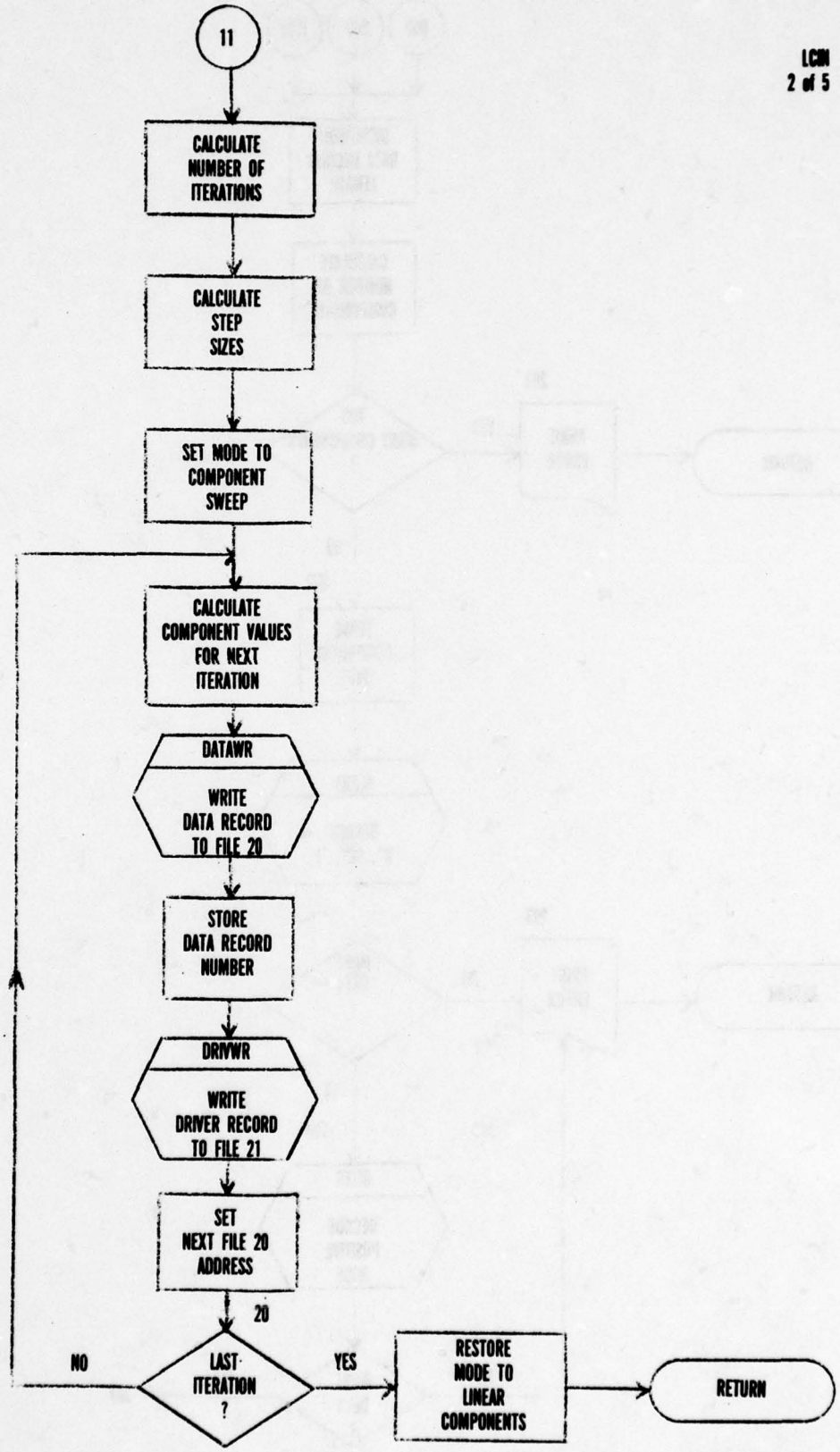
Otherwise the decoding, formatting, and storage technique

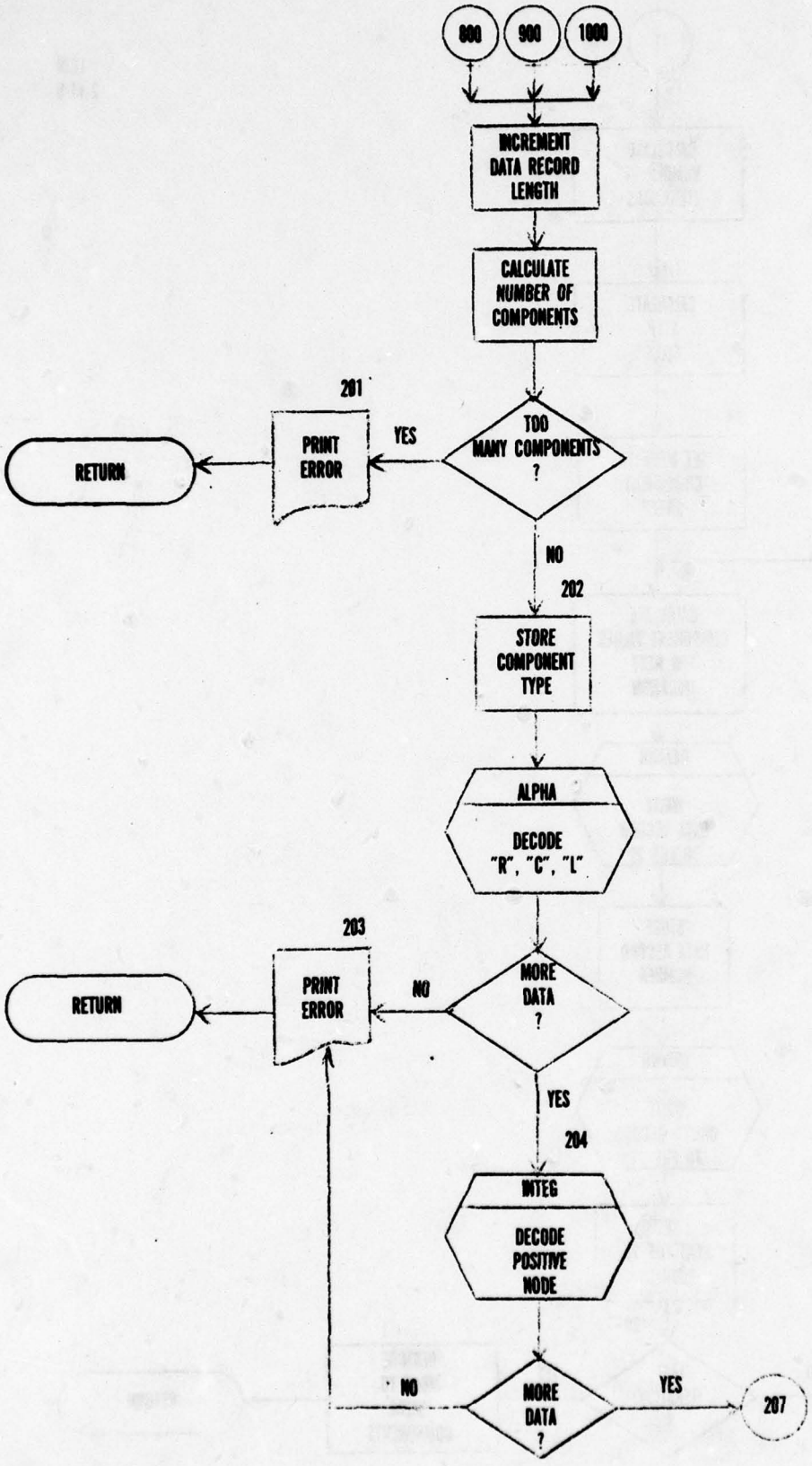
for AC-parameters is identical to that of B-parameters except that NXTAC is used for the parameter count.

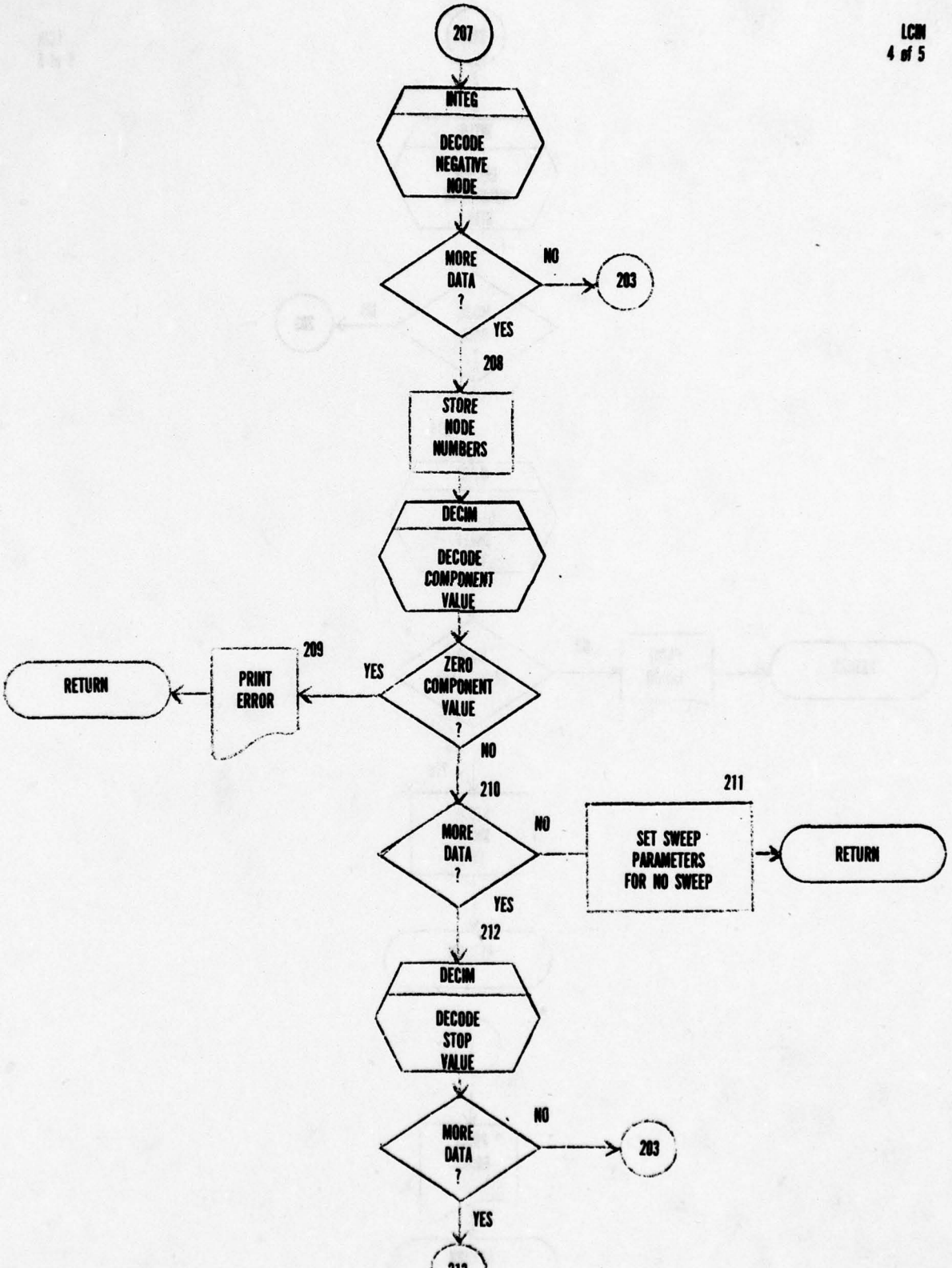
Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.

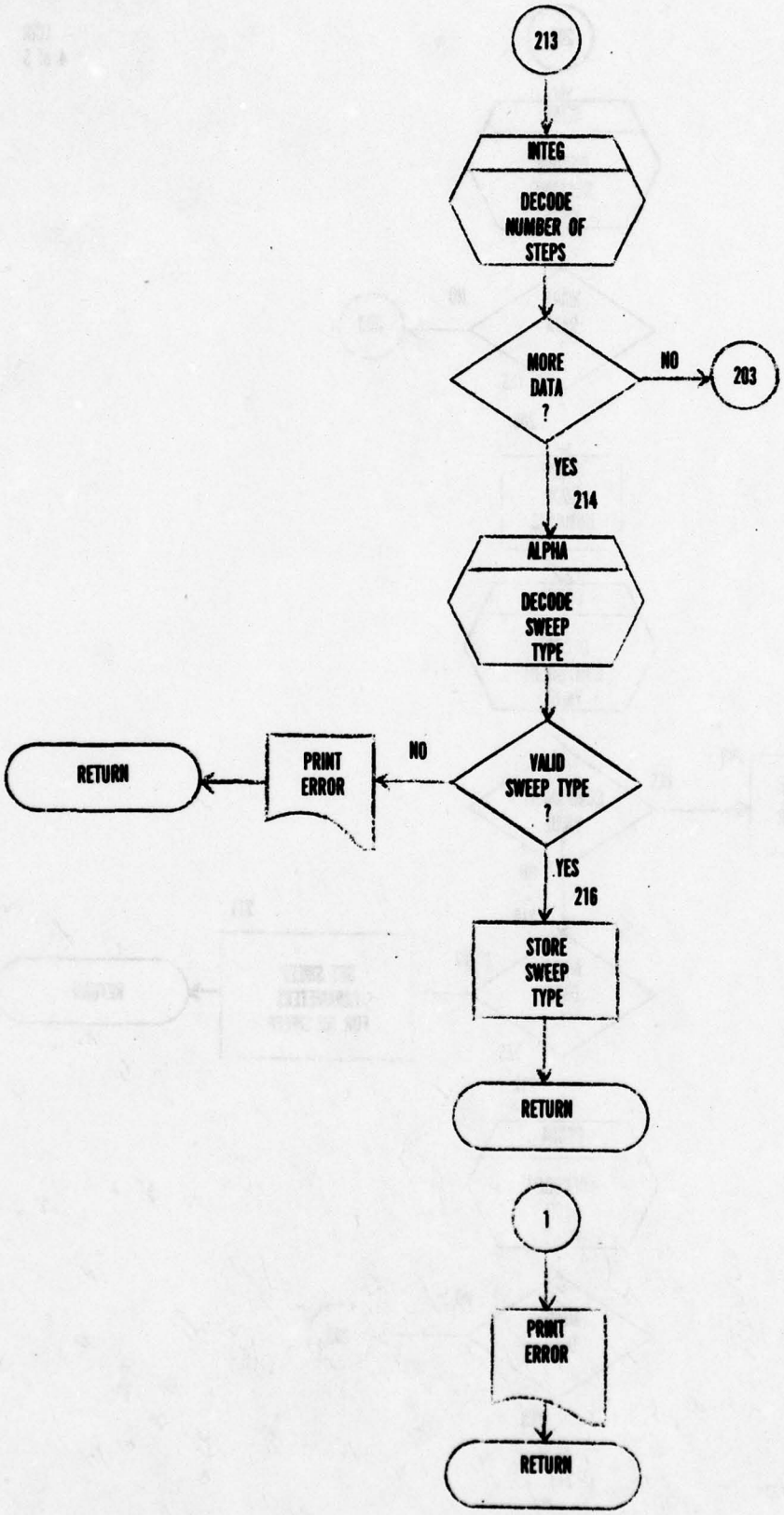

LCIN
2
ALPHA
SCAN
PACKR
PACKL
2
DATAWR
2
DRIVWR
2
DECIM
SCAN
PACKR
PACKL
6
ERROR
3
INTEG
SCAN
PACKR
PACKL
DRVOUT
DRIVWR
SHIFT
RSHIFT


START
TEST TYPE
100
1
800
8
900
9
1000
10
1
OTHER
100
101
PRINT ERROR
NO
ANY COMPONENTS INPUT ?
YES
102
103
RETURN
YES
INPUT ERRORS ?
NO
104
DATAWR
WRITE DATA RECORD TO FILE 20
STORE DATA RECORD NUMBER
DRVOUT
WRITE DRIVER RECORD TO FILE 21
RETURN
NO
SWEEP ?
YES
11


11
CALCULATE NUMBER OF ITERATIONS
CALCULATE STEP SIZES
SET MODE TO COMPONENT SWEEP
CALCULATE COMPONENT VALUES FOR NEXT ITERATION
DATAWR
WRITE DATA RECORD TO FILE 20
STORE DATA RECORD NUMBER
DRIVWR
WRITE DRIVER RECORD TO FILE 21
SET NEXT FILE 20 ADDRESS
20
LAST ITERATION ?
NO
YES
RESTORE MODE TO LINEAR COMPONENTS
RETURN


800
900
1000
INCREMENT DATA RECORD LENGTH
CALCULATE NUMBER OF COMPONENTS
201
TOO MANY COMPONENTS ?
YES
PRINT ERROR
RETURN
NO
202
STORE COMPONENT TYPE
ALPHA
DECODE "R", "C", "L"
203
MORE DATA ?
NO
PRINT ERROR
RETURN
YES
204
INTEG
DECODE POSITIVE NODE
MORE DATA ?
NO
YES
207


207
INTEG
DECODE NEGATIVE NODE
MORE DATA ?
NO
203
YES
208
STORE NODE NUMBERS
DECIM
DECODE COMPONENT VALUE
ZERO COMPONENT VALUE ?
YES
209
PRINT ERROR
RETURN
NO
210
MORE DATA ?
NO
211
SET SWEEP PARAMETERS FOR NO SWEEP
RETURN
YES
212
DECIM
DECODE STOP VALUE
MORE DATA ?
NO
203
YES
213


213
INTEG
DECODE NUMBER OF STEPS
MORE DATA ?
NO
203
YES
214
ALPHA
DECODE SWEEP TYPE
VALID SWEEP TYPE ?
NO
PRINT ERROR
RETURN
YES
216
STORE SWEEP TYPE
RETURN
1
PRINT ERROR
RETURN

NAME: LCIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *LINEAR COMPONENTS card sequence according to the value of TYPE:

Type = 1 End of card sequence

Type = 8 Resistor

Type = 9 Capacitor

Type = 10 Inductor

VARIABLES:

INC = Number of increments for sweep

ISW = End of data switch; Off = 0, On = 1

ITYPE = Sweep code; 0 = No sweep
1 = Linear
2 = Logarithmic

M = Intermediate storage of sweep type

NCOMP = Length of linear component data record in words

NIT = Maximum number of sweep iterations

NODEA = Positive node of connection

NODEB = Negative node of connection

NUM = Number of linear components this card sequence

STRING = Alphanumeric string representation of input

value

STEP = Sweep increment value

STOP = Sweep stop value

TYPES = Valid sweep types

SUBROUTINES CALLED:

ALPHA DATAWR DECIM DRIVWR DRVOUT

ERROR INTEG

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine LCIN is the input processor for LINEAR COMPONENTS card sequences. Its function is to decode the linear components input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Unlike most NCAP input card sequences which map to a single driver record and a data record of fixed length, linear components card sequences result in one or more driver records, each with an associated data record of variable length. The number of driver records generated is related to the number of linear component sweep iterations defined in the card sequence, while the length of each data record depends on the number of components defined in the card sequence.

Each call to subroutine LCIN causes a single NCAP input card

to be processed. The card image and its TYPE code are transmitted through common storage to subroutine LCIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

The closing of a linear components definition begins by testing the input for errors. If no linear component cards were processed (NCOMP.LE.Ø) or if the input error switch is on (JABT=1), return is made to the calling program with an appropriate error condition. Otherwise the data record is written to file 2Ø from the first NCOMP words of the common data buffer BUFF by subroutine DATAWR.

The linear components data record is structured as a series of NUM contiguous 4-word entries, one for each linear component defined in the card sequence:

BUFF(1)-(4)	First Component Definition
BUFF(5)-(8)	Second Component Definition
.	.
.	.
.	.
BUFF(NCOMP-3)-(NCOMP)	Last Component Definition

The first word of each entry contains an integer code representing the component type: 1 = Resistor, 2 = Capacitor, 3 = Inductor. The second and third words represent the positive and negative nodes of connection. The fourth word contains the

value of the component:

BUFF(1),(5),...(NCOMP-3)	Component Codes
BUFF(2),(6),...(NCOMP-2)	Positive Nodes of Connection
BUFF(3),(7),...(NCOMP-1)	Negative Nodes of Connection
BUFF(4),(8),...(NCOMP)	Component Values

The linear components driver record is written by subroutine DRVOUT to file 21 from the first ten words of global common, where:

MODE=4	Linear Component Identifier
STADD	Data File Record Number
LNGTH=NCOMP	Length of Data Record in Words
MISC(1)=NCOMP	Number of Linear Components x 4

Other MISC driver parameters are not used.

If component sweeping is not specified in the card sequence (i.e. INC(I)=0 for all I=1,NUM), subroutine LCIN returns to the calling program without further processing. Otherwise the sweep specifications are translated into appropriate driver and data records. First the number of sweep iterations, NIT, is derived from the largest INC parameter specified in the linear component card sequence. (Each of the NIT sweep iterations causes one driver and data record to be generated internally and appended to the disk files.)

Then a sweep increment STEP(I) is calculated for each of the NUM components in the card sequence. The increment is a function of the original component value stored at BUFF (I*4) and the

STOP, INC, and ITYPE parameters associated with the component as follows:

1) For ITYPE(I)=0, no sweep is assumed and STEP(I)=0.

2) For ITYPE(I)=1,linear sweep is indicated and:

$$STEP(I) = \frac{STOP(I)-BUFF(J)}{INC(I)-1.}$$

3) For ITYPE(I)=2, logarithmic sweep is indicated:

$$STEP(I) = \left[\frac{STOP(I)}{BUFF(J)}\right]^{\frac{1.}{INC(I)-1.}}$$

After the increments have been calculated the sweep driver and data records are created and appended to the disk files. Like standard driver records, sweep driver records are developed in the first ten words of global common. A component sweep data record consists of:

MODE=24	Component Sweep Identifier
STADD	Data File Record Number
LNGTH=NCOMP	Length of Data Record in Words
MISC(1)=NCOMP	Number of Components x 4

Sweep data records have the save structure as original linear component data records and are developed in the common data buffer BUFF by an iterative process. The first sweep data record is derived from the original linear component data record in BUFF by performing an incrementation operation on each component value according to its corresponding STEP, STOP, and

ITYPE parameters. The NUM component values in the buffer are incremented one-by-one under the control of the index J. Each component has a corresponding STEP(J), STOP(J), and ITYPE(J) which define the incrementation performed on its value stored at BUFF(J*4):

1) For ITYPE(J)=0, no sweep is indicated. No incrementation is performed and BUFF(J*4) remains constant at its original value.

2) For ITYPE(J)=1, linear sweep is indicated and the incrementation is additive. For K=J*4:

   BUFF(K)=BUFF(K)+STEP(J)

   unless STOP(J) is exceeded, in which case BUFF(K) remains constant at its last value.

3) For ITYPE(J)=2, logarithmic sweep is indicated and the incrementation is multiplicative. For K=J*4:

   BUFF(K)=BUFF(K)*STEP(J)

   If STOP(J) is exceeded, BUFF(K) remains constant at STOP(J).

After every component value in the buffer has been updated, the new data record is written to file 20 at record number DATREC by subroutine DATAWR. Then subroutine DRIVWR is called to write the sweep driver record to file 21. The data record number STADD is updated and program control transfers back to process the next sweep iteration. The creation of sweep records continues under the control of the index I until all NIT sweep iterations have been translated to data and driver records. Subroutine LCIN returns to the calling program after restoring MODE to 4.

TYPE=8, 9, or 10 Resistor, Capacitor, or Inductor Card

The data record length NCOMP is incremented to accommodate a new four-word component definition and the component count NUM is calculated. If more then 50 linear components have ben processed in the card sequence (NUM.GT.50), subroutine LCIN exits with an appropriate error condition. Otherwise the new component code is set to TYPE-7 and stored in BUFF(NCOMP-3).

Subroutine LCIN anticipates the input of a component identifier, two nodes of connection, and a component value for each component in the card sequence. Furthermore, for element sweeping, three sweep parameters are also expected. During decoding of the input card, if an end-of-data is encountered (ISW=1) before all the required parameters have been processed, subroutine LCIN exits with an appropriate error condition.

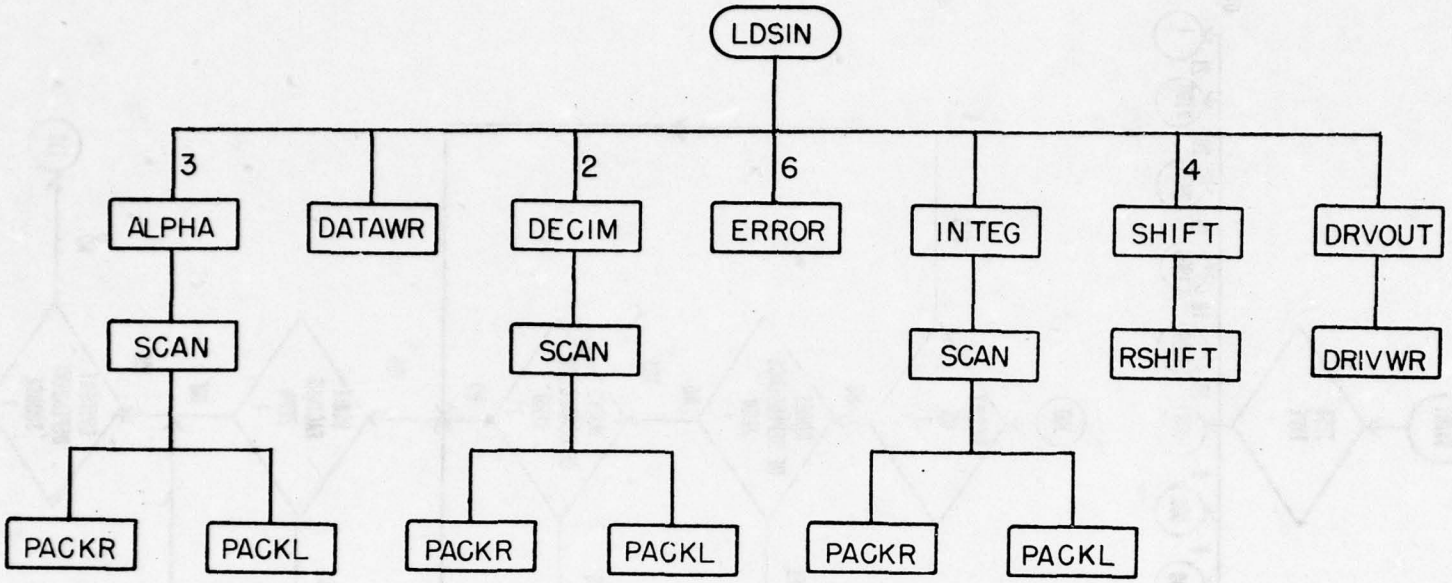
The R, C, or L identifier is decoded by subroutine ALPHA. The nodes of connection are decoded and formatted by subroutine INTEG and stored in BUFF(NCOMP-2) and BUFF(NCOMP-1). The component value is decoded and formatted by subroutine DECIM and stored in BUFF(NCOMP). If the component value is zero, subroutine LCIN exits with an appropriate error condition.

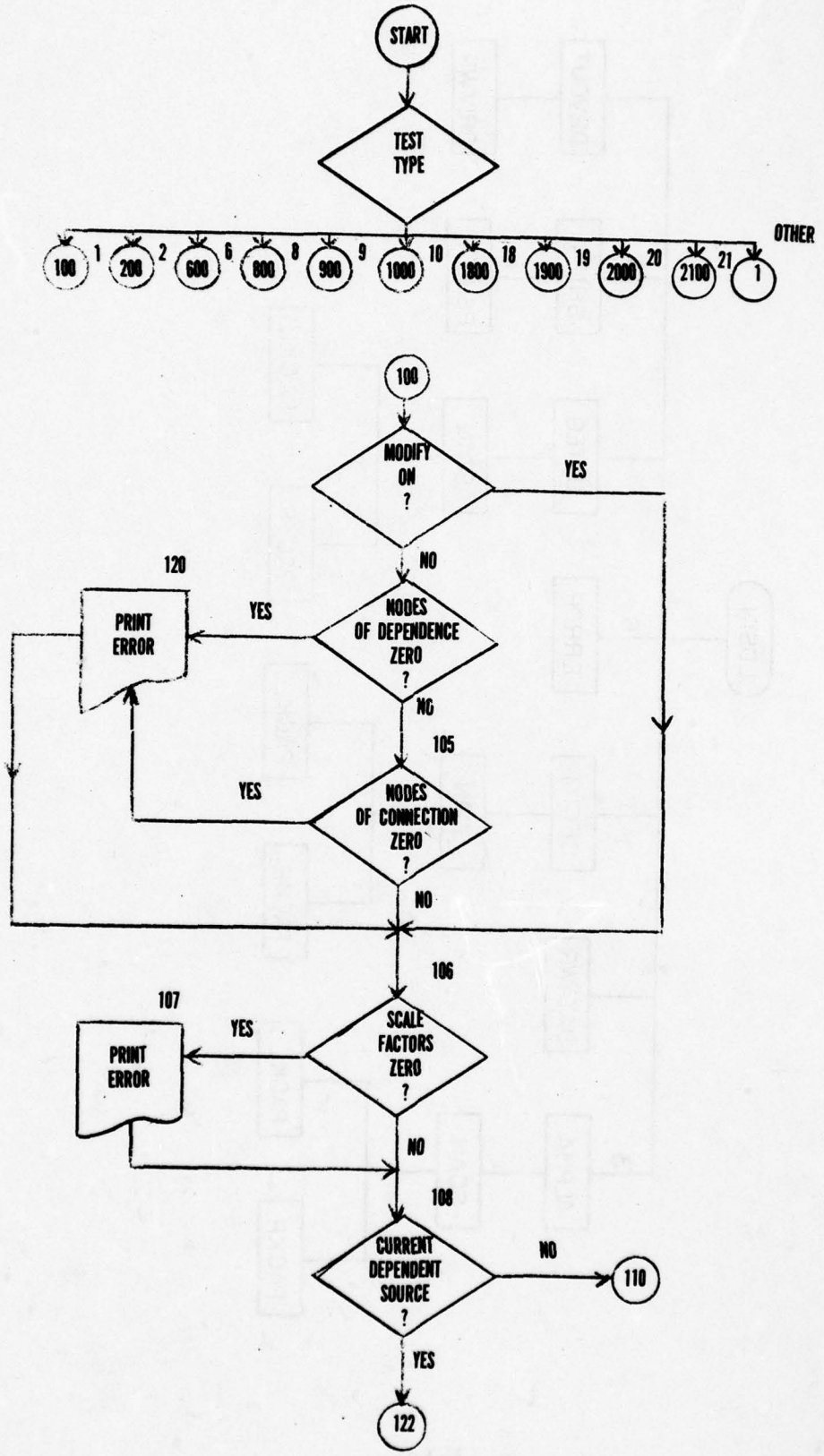
At this point ISW is used to determine the presence of component sweep parameters on the card. If the end-of-data switch is on (ISW=1), the subroutine assumes no sweeping and exits after disabling the sweep parameters ITYPE and INC.

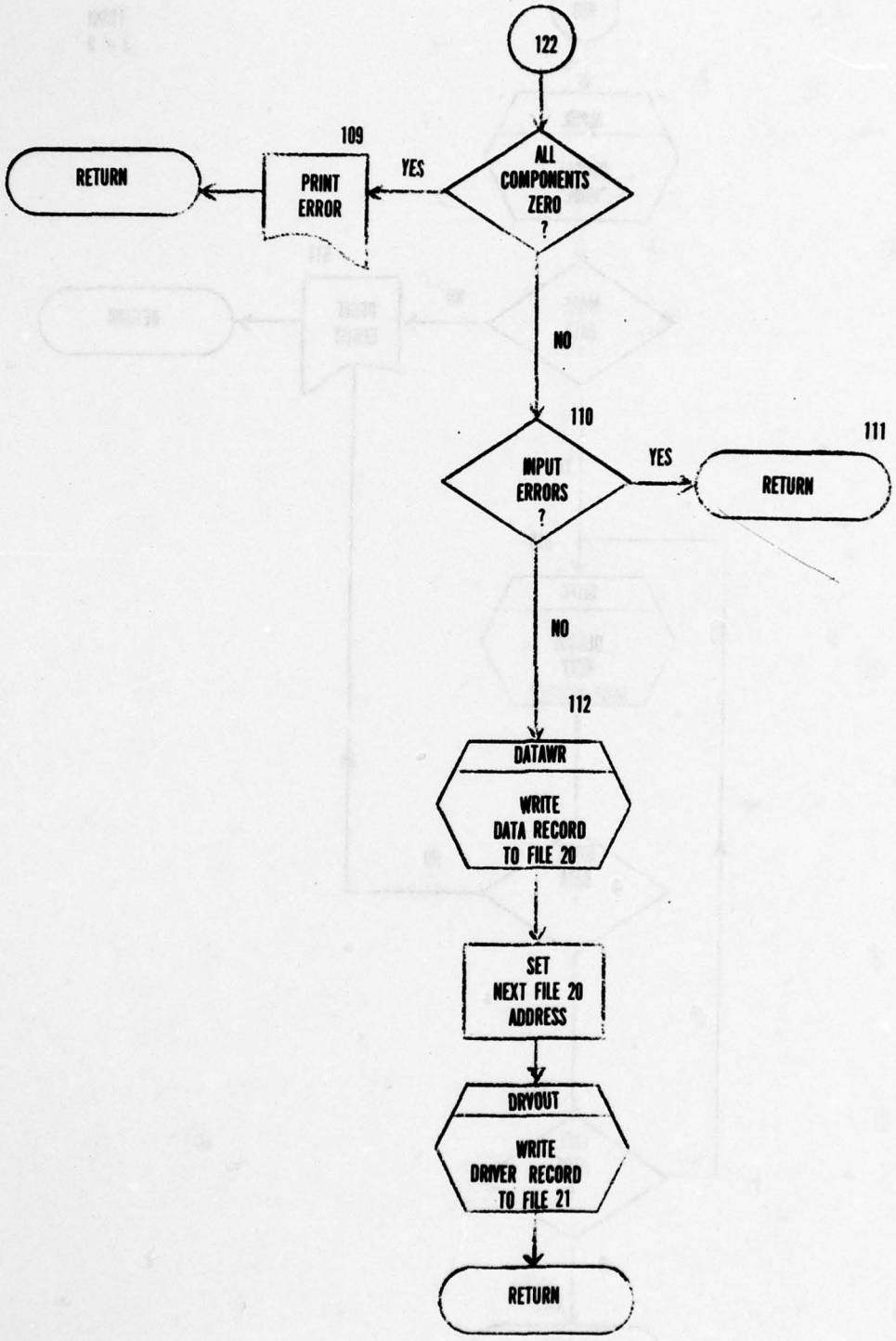
Otherwise the STOP, INC, and ITYPE parameters are decoded and formatted by subroutine DECIM, INTEG, and ALPHA respectively.

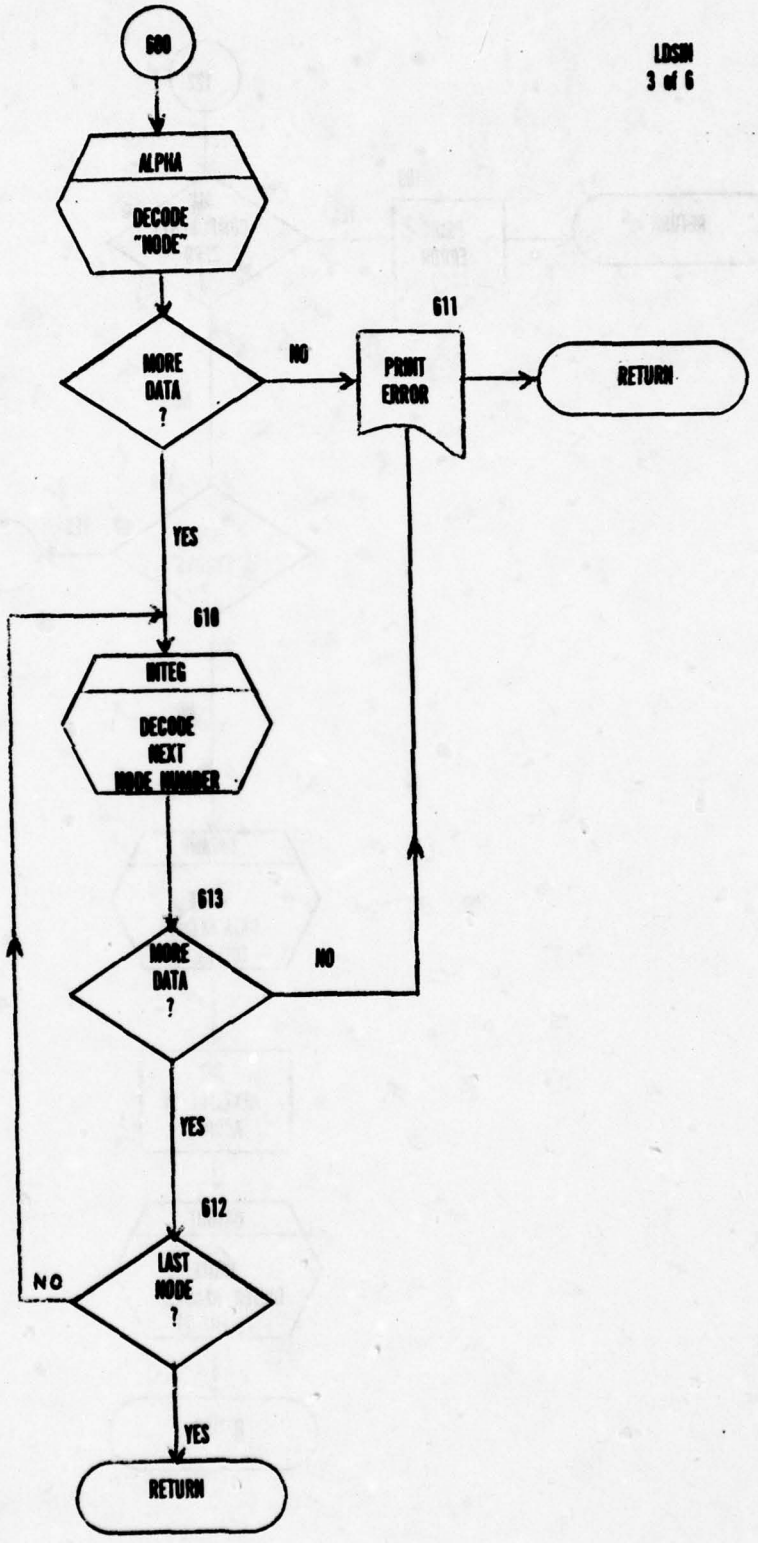
The sweep type is tested against a table of valid alphanumeric types. If a match is found the sweep type is encoded in ITYPE(NUM) and subroutine LCIN returns to the calling program. If no match is found, subroutine LCIN exits with an appropriate error condition.

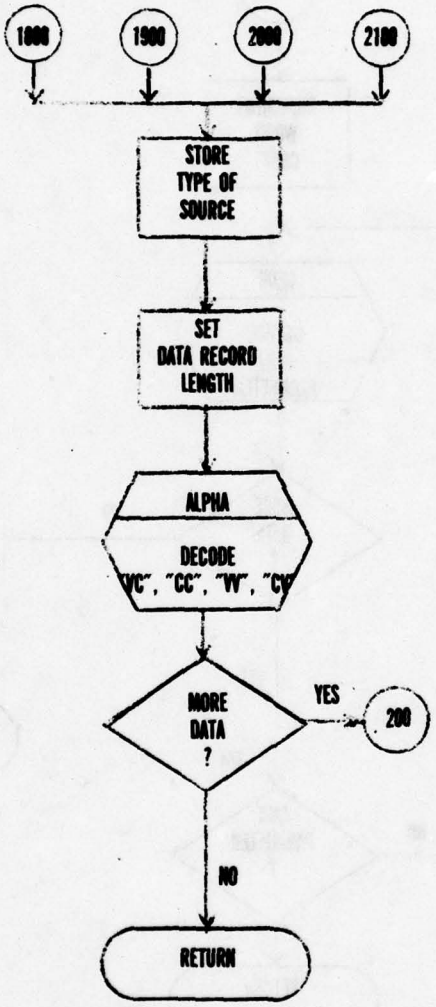
Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.

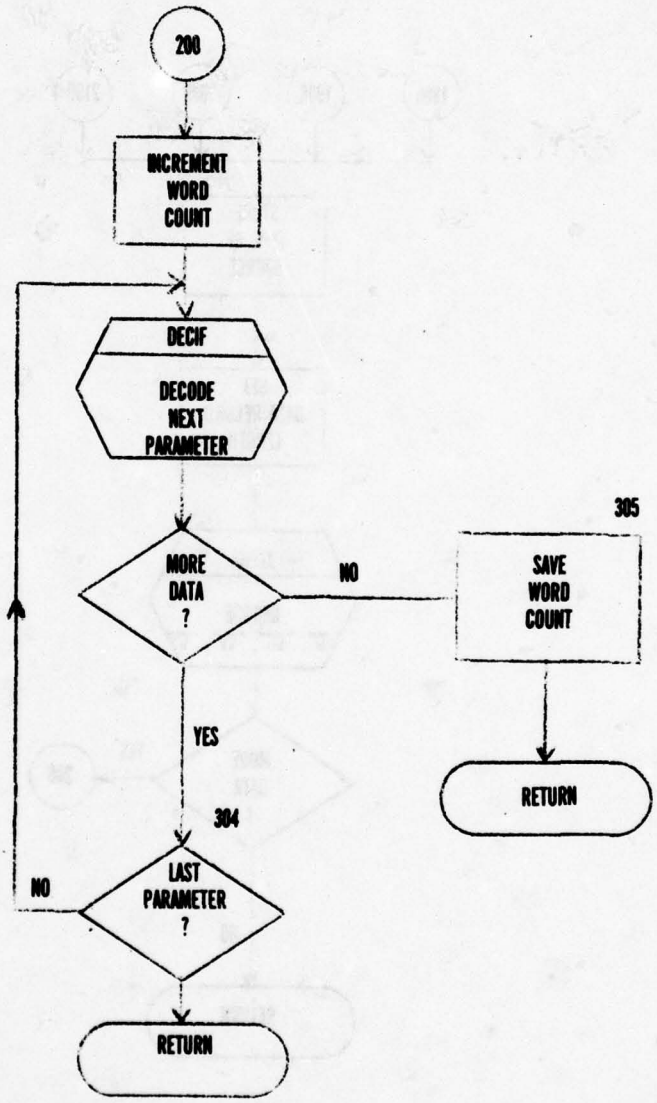

LDSIN
3
ALPHA
DATAWR
2
DECIM
6
ERROR
INTEG
4
SHIFT
DRVOUT
SCAN
SCAN
SCAN
RSHIFT
DRIVWR
PACKR
PACKL
PACKR
PACKL
PACKR
PACKL

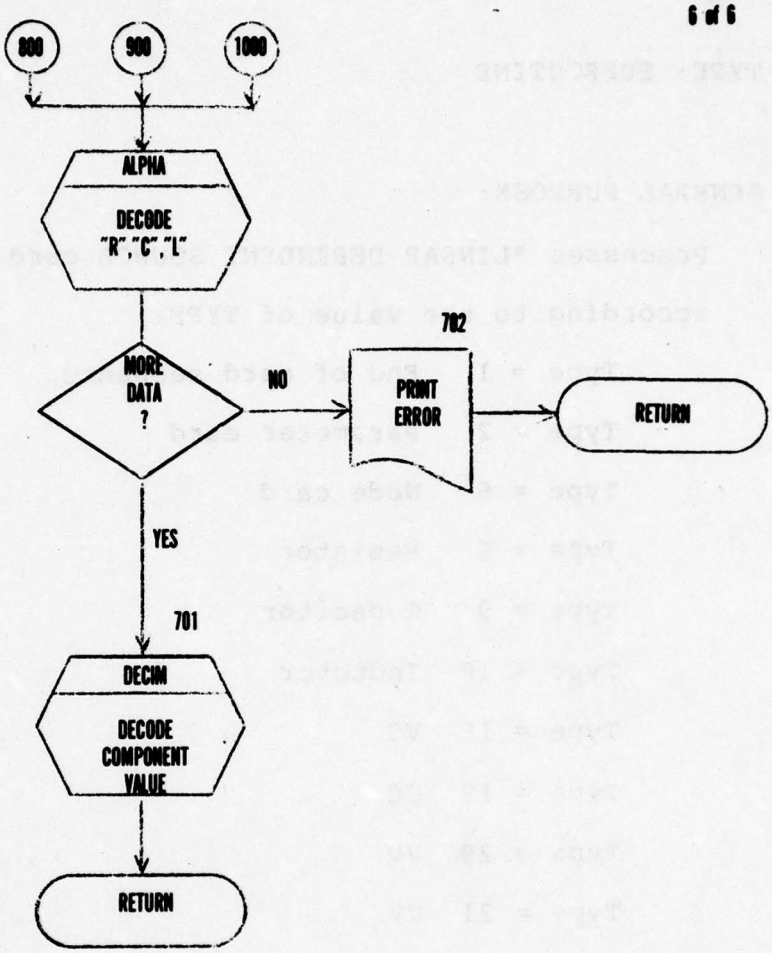

START
TEST TYPE
OTHER
100
1
200
2
600
6
800
8
900
9
1000
10
1800
18
1900
19
2000
20
2100
21
1
100
MODIFY ON ?
YES
NO
120
PRINT ERROR
YES
NODES OF DEPENDENCE ZERO ?
NO
105
YES
NODES OF CONNECTION ZERO ?
NO
106
107
PRINT ERROR
YES
SCALE FACTORS ZERO ?
NO
108
CURRENT DEPENDENT SOURCE ?
NO
110
YES
122

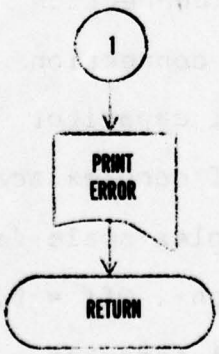

122
ALL COMPONENTS ZERO ?
YES
109
PRINT ERROR
RETURN
NO
110
INPUT ERRORS ?
YES
111
RETURN
NO
112
DATAWR
WRITE DATA RECORD TO FILE 20
SET NEXT FILE 20 ADDRESS
DRVOUT
WRITE DRIVER RECORD TO FILE 21
RETURN


600
ALPHA
DECODE
"MODE"
MORE
DATA
?
NO
611
PRINT
ERROR
RETURN
YES
610
INTEG
DECODE
NEXT
MODE NUMBER
613
MORE
DATA
?
NO
YES
612
LAST
MODE
?
NO
YES
RETURN


1800
1900
2000
2100
STORE TYPE OF SOURCE
SET DATA RECORD LENGTH
ALPHA
DECODE
"VC", "CC", "W", "CV
MORE DATA ?
YES
200
NO
RETURN


200
INCREMENT WORD COUNT
DECIF
DECODE NEXT PARAMETER
MORE DATA ?
NO
305
SAVE WORD COUNT
RETURN
YES
304
LAST PARAMETER ?
NO
RETURN


800
900
1000
ALPHA
DECODE
"R","G","L"
MORE
DATA
?
NO
702
PRINT
ERROR
RETURN
YES
701
DECIM
DECODE
COMPONENT
VALUE
RETURN


1
PRINT
ERROR
RETURN

NAME: LDSIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *LINEAR DEPENDENT SOURCE card sequence according to the value of TYPE:

Type = 1	End of card sequence
Type = 2	Parameter card
Type = 6	Node card
Type = 8	Resistor
Type = 9	Capacitor
Type = 10	Inductor
Type = 18	VC
Type = 19	CC
Type = 20	VV
Type = 21	CV

VARIABLES:

A	= Positive node of connection
B	= Negative node of connection
C	= Value of parallel capacitor
CI	= Imaginary part of complex scale factor
CR	= Real part of complex scale factor
ISW	= End of data switch: Off = 0, On = 1
L	= Value of parallel inductor
NW	= Word counter for parameter input

R = Value of parallel resistor

STRING = Alphanumeric string representation of input value

SOURCE = Type of source· 1 = VC
2 = CC
3 = VV
4 = CV

XX = Positive node of dependence

YY = Negative node of dependence

SUBROUTINES CALLED·

ALPHA DATAWR DECIM DRIVWR

ERROR INTEC

CALLING PROGRAMS·

PRCSS

DESCRIPTION:

Subroutine LDSIN is the input processor for LINEAR DEPENDENT SOURCE card sequences. Its function is to decode the linear dependent source input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine LDSIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine LDSIN which processes the card according to its TYPE as follows·

TYPE=1 End of Card Sequence

The closing of a linear dependent source definition begins by testing the input for errors. Subroutine LDSIN exits with an appropriate error message if any of the following input errors are detected:

1) All four nodes (A, B, XX, and YY) are zero and the card sequence is not a modify function (MODFY=0)
2) The complex scale factor (CR, CI) is zero
3) For current controlled sources, all three parallel components (R, C, and L) are zero
4) The input error switch is on (JABT=1)

Otherwise the linear dependent source data record is written to file 20 from the common data buffer BUFF and the file 20 record number DATREC is updated to point beyond the new data record.

Because the linear dependent sources do not transmit any internally generated data between NCAP phases, their data records contain only input values. The length of a linear dependent source data record depends on the type of source specified: voltage controlled sources require 2 words of data storage, while current controlled sources require 5 words.

After the data record has been written to disk, subroutine DRVOUT is called to complete the driver record definition and write it to file 21. The linear dependent source driver record is developed in the first ten words of global common and consists

of·

MODE=25	Linear Dependent Source Identifier
STADD	Data File Record Number
LNGTH=2 or 5	Length of Data Record in Words
MISC(1)=SOURCE	Source Code: 1=VC, 2=CC, 3=VV, 4=CV.
MISC(3)=XX	Positive Node of Dependence
MISC(4)=YY	Negative Node of Dependence
MISC(5)=A	Positive Node of Connection
MISC(6)=B	Negative Node of Connection

After returning from subroutine DRVOUT, subroutine LDSIN returns to the calling program.

TYPE=2 Parameter Card

One or two decimal input values are extracted from the card image, formatted and stored in the data buffer. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of values input on previous data cards. Parameter storage begins at BUFF(NW) where NW=NXT+1 and proceeds through successive locations as data values are decoded by subroutine DECIM. If an end-of-data is encountered (ISW=1) before BUFF(2) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation of additional

parameter cards.

TYPE=6 Node Card

The NODE identifier is decoded by subroutine ALPHA. Then the nodes of dependence and nodes of connection are extracted from the card image and formatted by subroutine INTEG and stored in MISC(3) through MISC(6). If an end-of-data is encountered (ISW=1) before four node numbers have been decoded, subroutine LDSIN exits with an appropriate error condition.

TYPE= 8, 9, or 10 Resistor, Capacitor, or Inductor Card

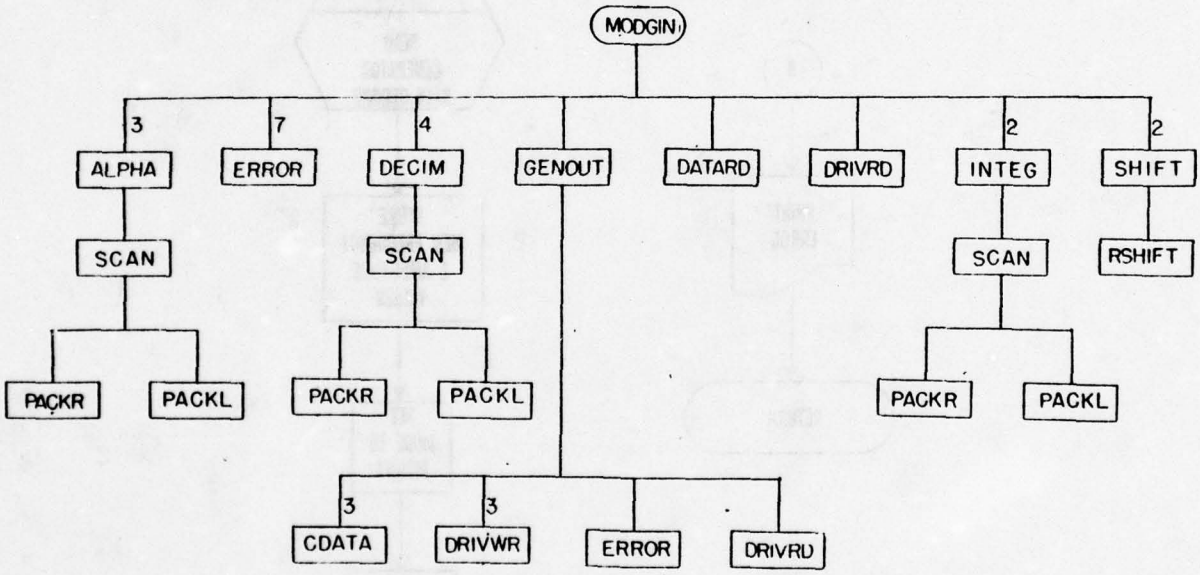
The R, C, or L identifier is decoded by subroutine ALPHA. If an end-of-data is encountered (ISW=1), subroutine LDSIN exits with an appropriate error condition. Otherwise the component value is extracted from the card image, formatted and stored at BUFF(TYPE-5) by subroutine DECIM.

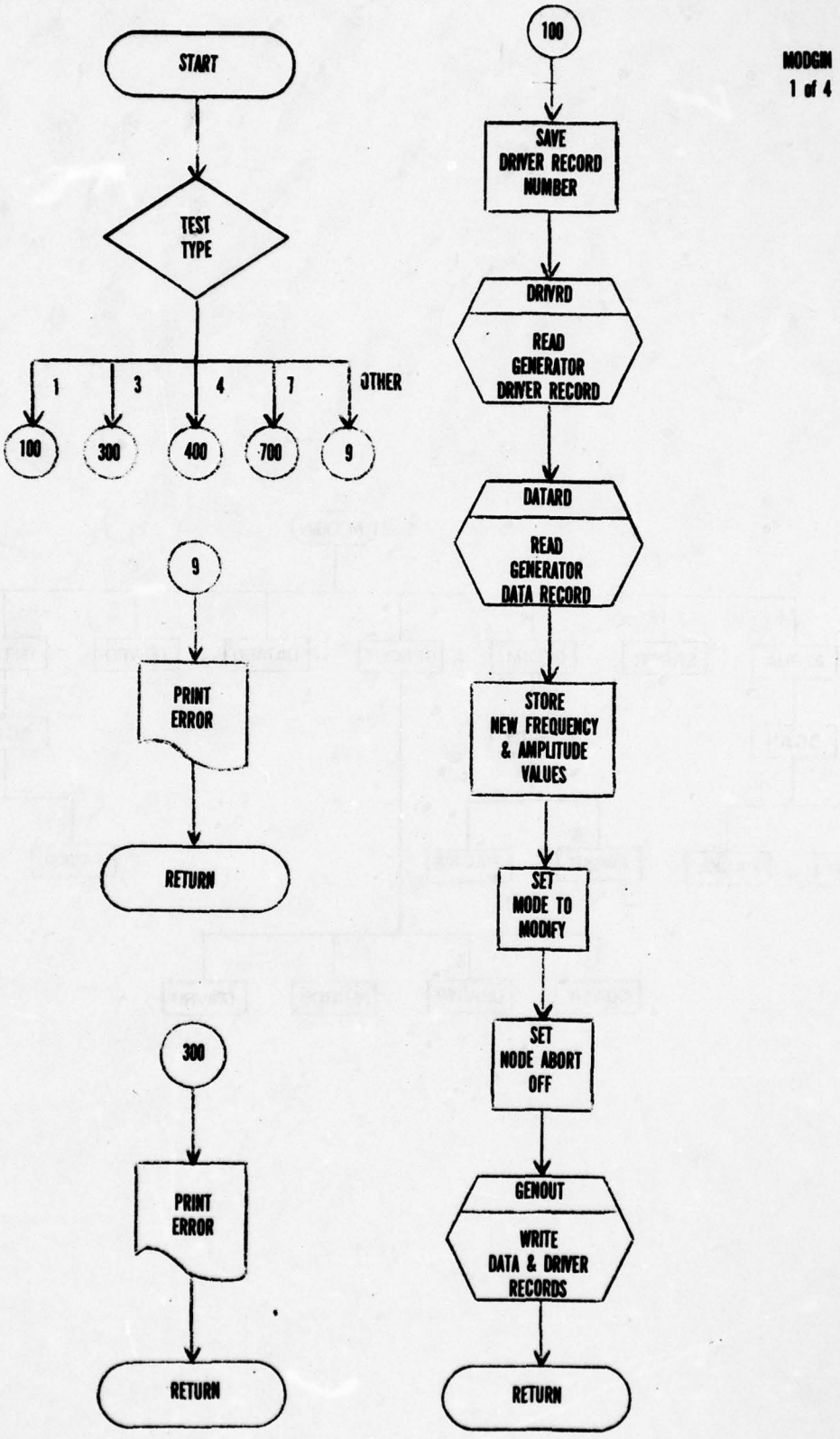
TYPE= 18, 19, 20, or 21 VC, CC, VV, or CV Card

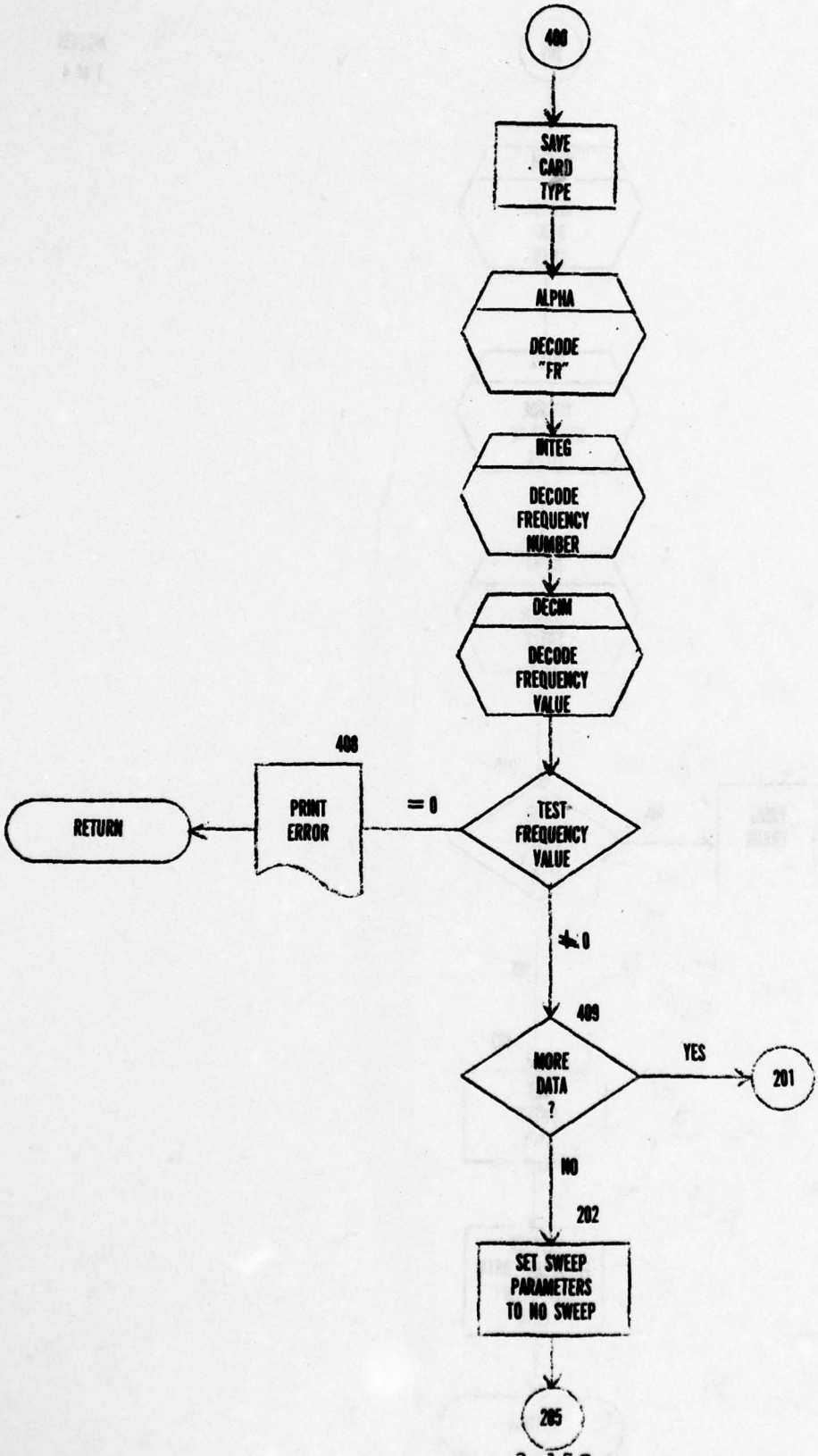
The SOURCE type is calculated according to TYPE=17 and the data record length is set: LNGTH=2 for voltage controlled sources, LNGTH=5 for current controlled sources. The VC, CC, VV,

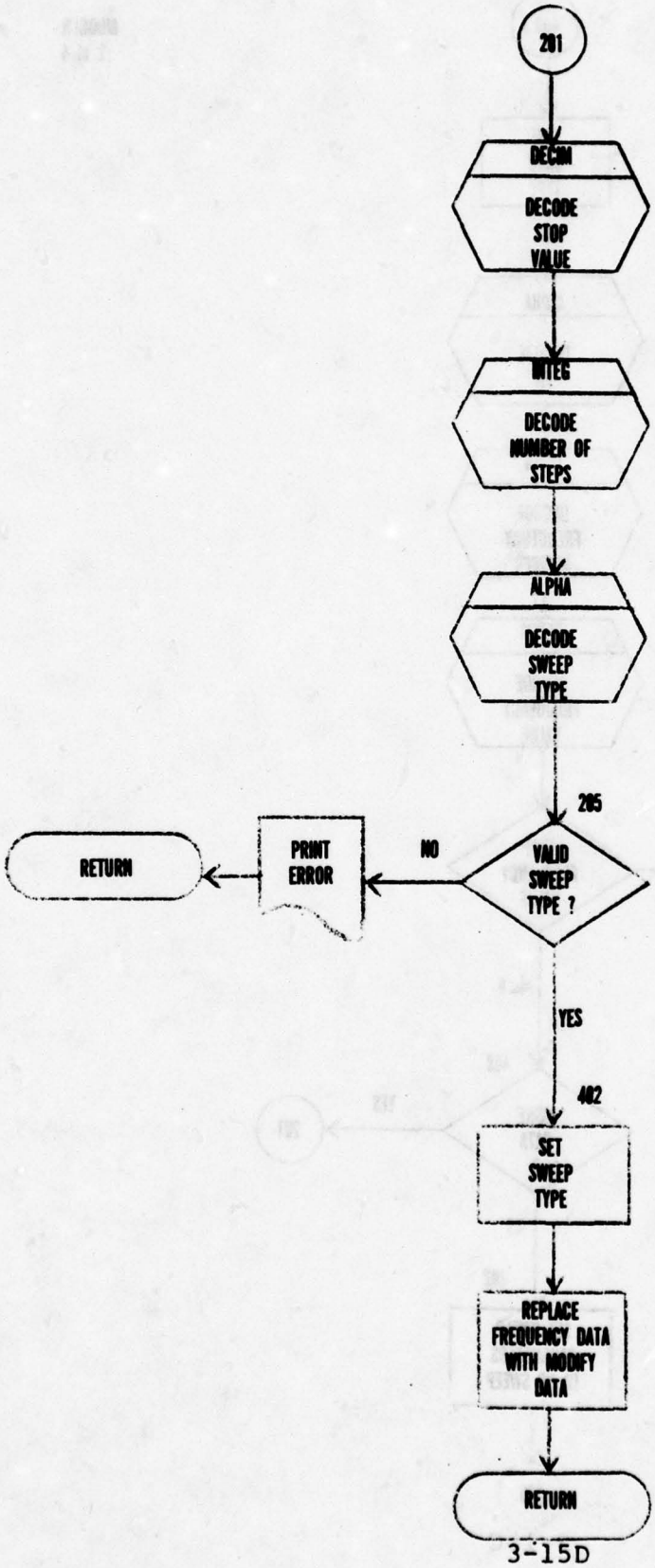
or CV identifier is decoded by subroutine ALPHA. If an end-of-data is encountered (ISW=1), subroutine LDSIN returns to the calling program. Otherwise control transfers to TYPE=2 processing for the remaining data on the card.

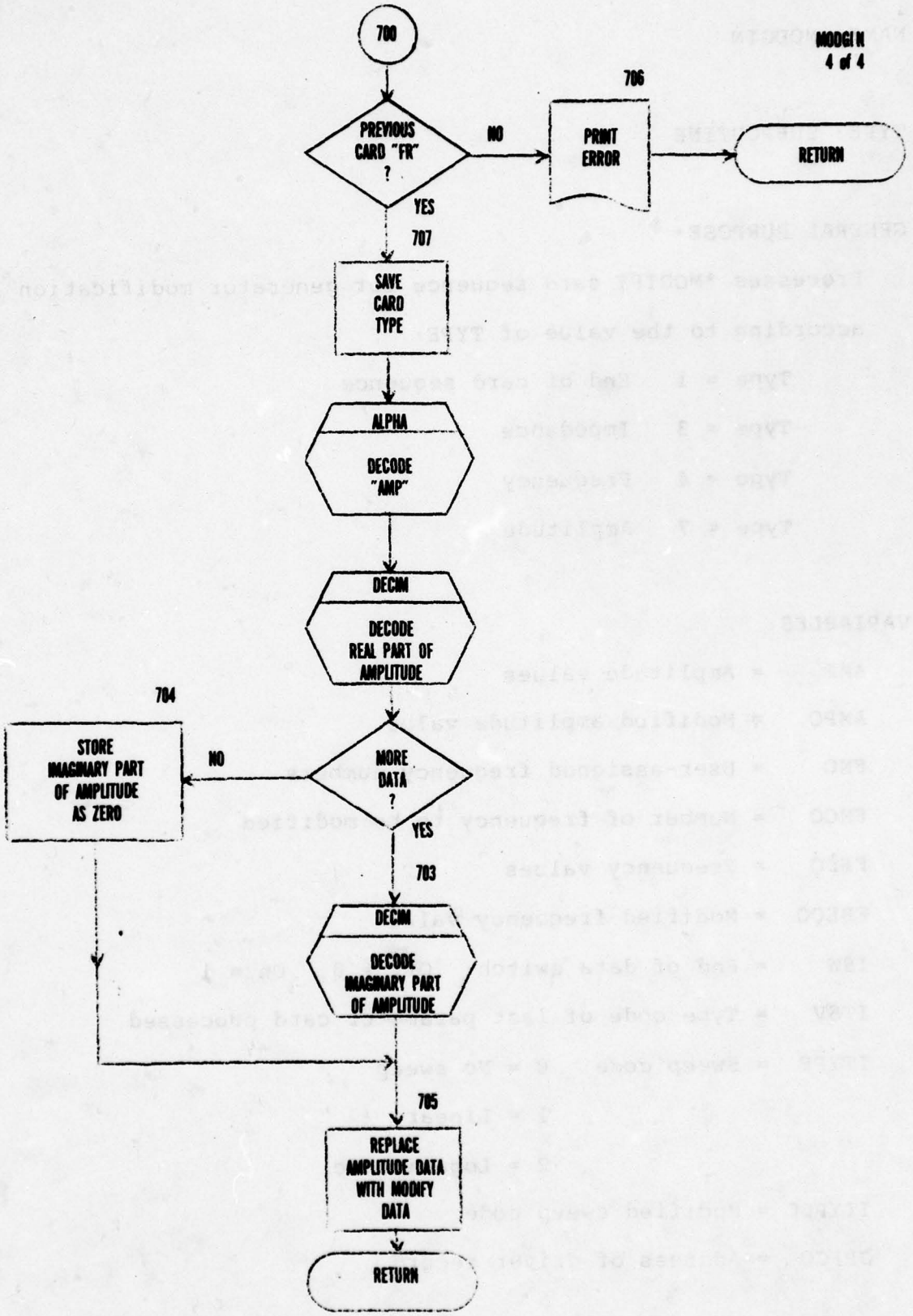
Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


MODGIN
3
ALPHA
SCAN
PACKR
PACKL
7
ERROR
4
DECIM
SCAN
PACKR
PACKL
GENOUT
3
CDATA
3
DRIVWR
ERROR
DRIVRD
DATARD
DRIVRD
2
INTEG
SCAN
PACKR
PACKL
2
SHIFT
RSHIFT


START
TEST TYPE
1
3
4
7
OTHER
100
300
400
700
9
9
PRINT ERROR
RETURN
300
PRINT ERROR
RETURN
100
SAVE DRIVER RECORD NUMBER
DRIVRD
READ GENERATOR DRIVER RECORD
DATARD
READ GENERATOR DATA RECORD
STORE NEW FREQUENCY & AMPLITUDE VALUES
SET MODE TO MODIFY
SET MODE ABORT OFF
GENOUT
WRITE DATA & DRIVER RECORDS
RETURN


400
SAVE CARD TYPE
ALPHA
DECODE "FR"
INTEG
DECODE FREQUENCY NUMBER
DECIM
DECODE FREQUENCY VALUE
408
TEST FREQUENCY VALUE
= 0
PRINT ERROR
RETURN
≠ 0
409
MORE DATA ?
YES
201
NO
202
SET SWEEP PARAMETERS TO NO SWEEP
205


201
DECIM
DECODE STOP VALUE
INTEG
DECODE NUMBER OF STEPS
ALPHA
DECODE SWEEP TYPE
205
VALID SWEEP TYPE ?
NO
PRINT ERROR
RETURN
YES
402
SET SWEEP TYPE
REPLACE FREQUENCY DATA WITH MODIFY DATA
RETURN


700
706
PREVIOUS CARD "FR" ?
NO
PRINT ERROR
RETURN
YES
707
SAVE CARD TYPE
ALPHA
DECODE "AMP"
DECIM
DECODE REAL PART OF AMPLITUDE
704
STORE IMAGINARY PART OF AMPLITUDE AS ZERO
NO
MORE DATA ?
YES
703
DECIM
DECODE IMAGINARY PART OF AMPLITUDE
705
REPLACE AMPLITUDE DATA WITH MODIFY DATA
RETURN

NAME· MODGIN

TYPE· SUBROUTINE

GENERAL PURPOSE·

Processes *MODIFY card sequence for generator modification according to the value of TYPE·

Type = 1 · End of card sequence

Type = 3 Impedance

Type = 4 Frequency

Type = 7 Amplitude

VARIABLES·

AMP = Amplitude values

AMPO = Modified amplitude value

FNO = User-assigned frequency numbers

FNOO = Number of frequency to be modified

FREQ = Frequency values

FREQO = Modified frequency value

ISW = End of data switch· Off = 0, On = 1

ITSV = Type code of last parameter card processed

ITYPE = Sweep code· 0 = No sweep

1 = Linear

2 = Logarithmic

ITYPEO = Modified sweep code

JRECO = Address of driver record

NFREQ = Number of frequencies this generator

NIT = Number of sweep iterations in generator being modified

STRING = Alphanumeric string representation of input value

STOP = Stop value for sweep

STOPC = Modified stop value

TYPES = Valid frequency sweep types

SUBROUTINES CALLED·

ALPHA DATARD DECIM DRIVRD ERROR

GENOUT INTEG

CALLING PROGRAMS·

PRCSS

DESCRIPTION·

Subroutine MODGIN is the input processor for MODIFY GENERATOR card sequences. Its function is to decode the generator modification input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine MODGIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine MODGIN which processes the card according to its TYPE as follows·

TYPE=1 End of Card Sequence

The modify generator data record is created by replacing the frequency and amplitude values of the previous generator record with the updated FREQ and AMP values processed by subroutine MODGIN. After saving the present driver record number in JRECO, the previous generator driver record is read by subroutine DRIVRD (its record number is obtained by backspacing over the NIT frequency sweep driver records which are stored between it and the present driver record at DRVREC). Using the record number STADD, the previous generator data is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF.

The updated frequency values are stored in BUFF(1)-(10) and the updated amplitudes in BUFF(11)-(30). MODE is set to 21 to identify the generator modification function, the node card switch is disabled, and subroutine GENOUT is called to write the data and driver records to disk.

TYPE=3 Impedance Card

Since generator impedances cannot be modified, any impedance card in a modify generator card sequence constitutes an input error which is handled by subroutine ERROR before exit to the calling program.

TYPE=4 Frequency Card

The card type code is saved in ITSV and the FR identifier is decoded by subroutine ALPHA. The frequency number is decoded and formatted by subroutine INTEG and stored in FNOO. The frequency value is decoded and formatted by subroutine DECIM and stored in FREQO. If the frequency value is zero, subroutine MODGIN exits with an appropriate error condition.

At this point the end-of-data switch ISW is used to determine the presence of frequency sweep parameters on the card. If the end-of-data in on (ISW=1), the parameters ITYPEO and INCO are set to indicate no sweep. Otherwise the sweep parameters STOPO, INCO, and ITYPEO are decoded and formatted by subroutines DECIM, INTEG, and ALPHA respectively.

The sweep type is tested against a table of valid alphanumeric types. If no match is found, subroutine MODGIN exits with an appropriate error condition. If a match is found, the sweep type is encoded in ITYPEO.

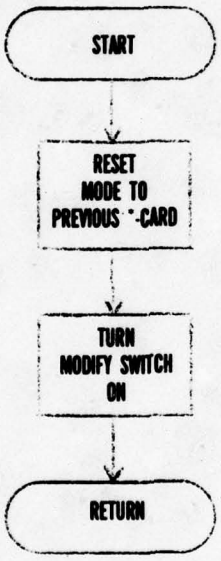
The number of the frequency being modified FNOO is compared against the frequency numbers defined for the previous generator FNO(I), I=1,10. If no match is found, subroutine MODGIN exits with an appropriate error condition. If a match is found, the table index I serves as the index for replacing FREQ(I), STOP(I), INC(I), and ITYPE(I) with the modified values, after which subroutine MODGIN returns to the calling program.

TYPE=7 Amplitude Card

In order for amplitude data to be associated with the proper frequency, each amplitude card must physically follow a frequency card in the input deck. Therefore if the last card processed was not a frequency card (ITSV $\neq$ 4), subroutine MODGIN exists with an appropriate error condition.

Otherwise the card type code is saved in ITSV and the AMP identifier is decoded by subroutine ALPHA. The real and imaginary parts of the complex amplitude are decoded and formatted by subroutine DECIM and stored in AMPO(1) and AMPO(2). The number of the frequency being modified FNOO is compared against the frequency numbers defined for the previous generator FNO(I), I=1,10. If no match is found, subroutine MODGIN exits with an appropriate error condition. If a match is found, the table index I is used to replace the original amplitude data AMP(1,I) and AMP(2,I) with the modified amplitude, after which subroutine MODGIN returns to the calling program.

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


START
RESET
MODE TO
PREVIOUS *-CARD
TURN
MODIFY SWITCH
ON
RETURN

NAME: MODIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Process *MODIFY card, resets MODE to identify device being modified, turns modify switch 'ON' so remainder of card sequence is processed according to MODE of device being modified

VARIABLES:

NONE

SUBROUTINES CALLED:

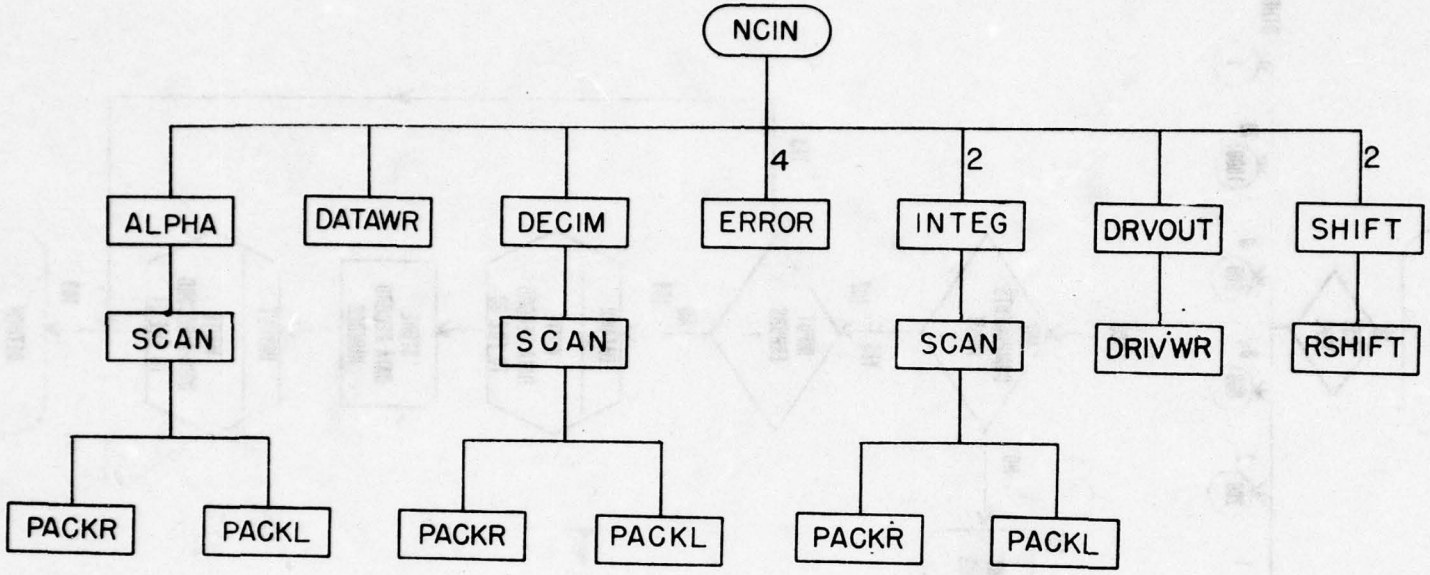
NONE

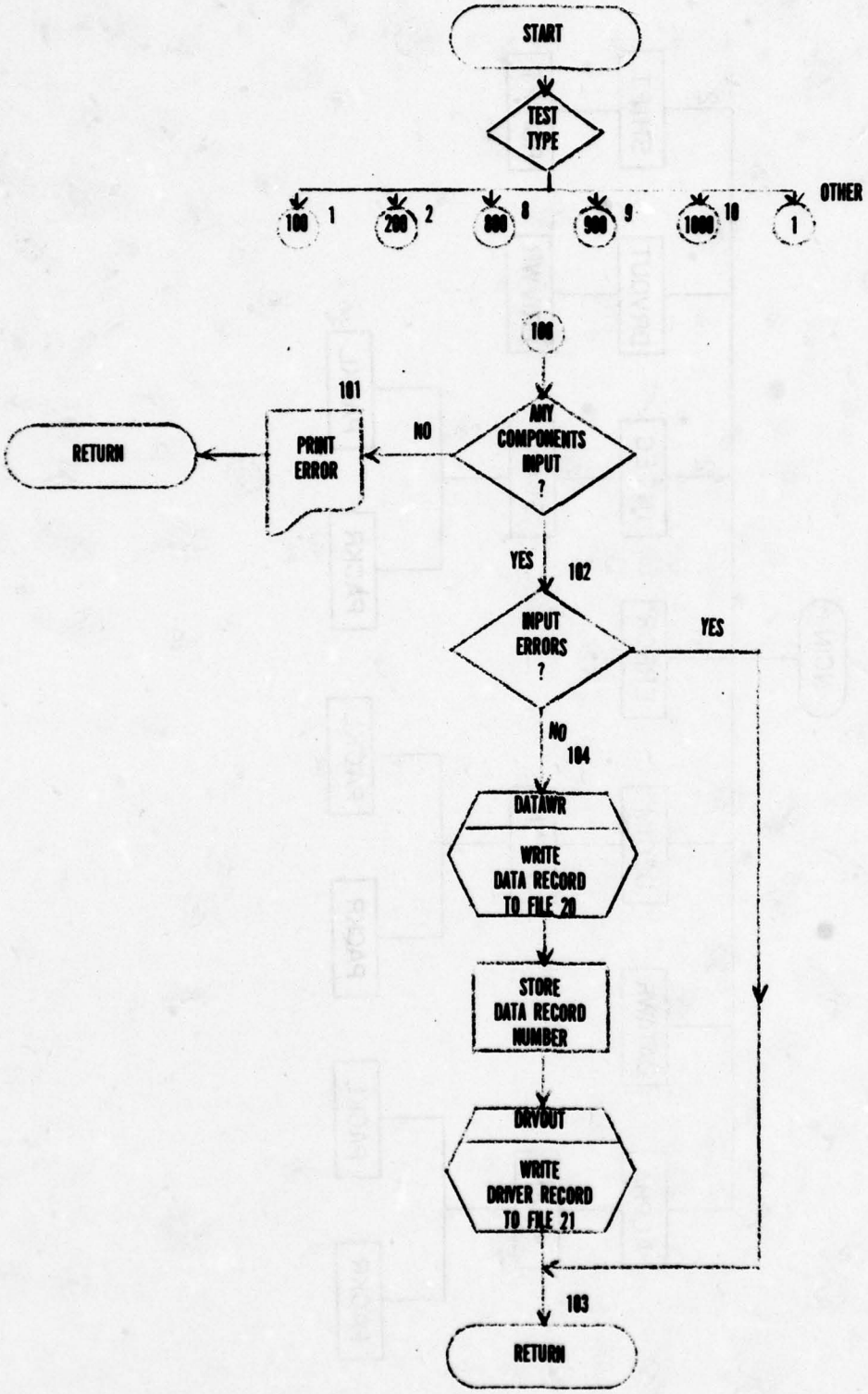
CALLING PROGRAMS:

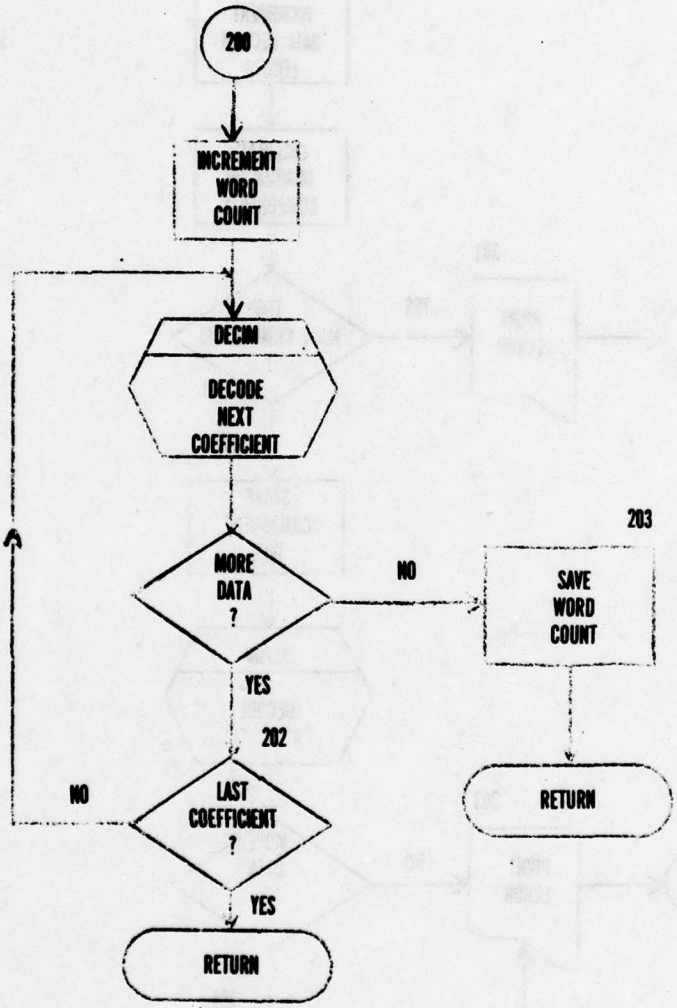
PRCSS

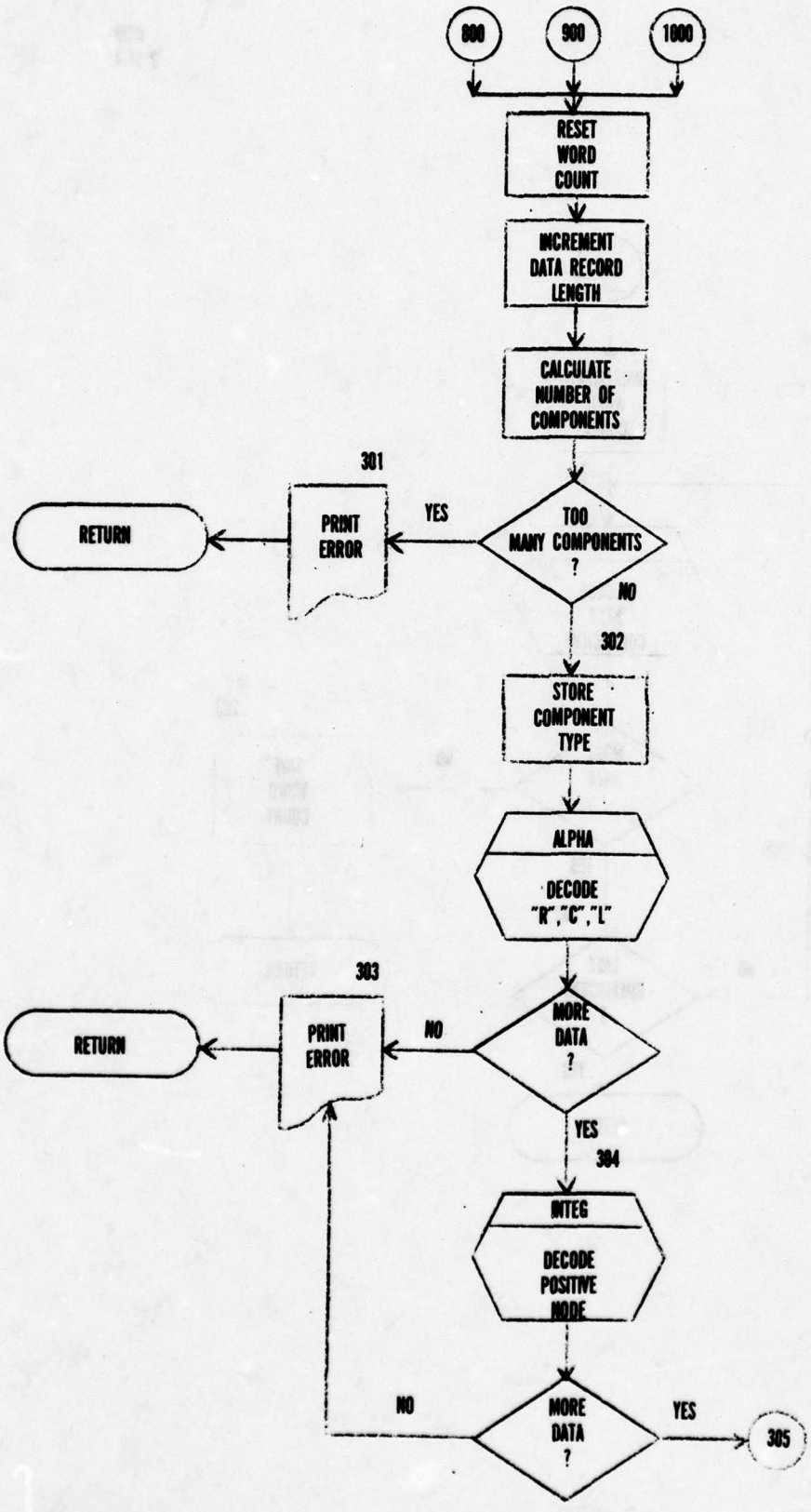
DESCRIPTION:

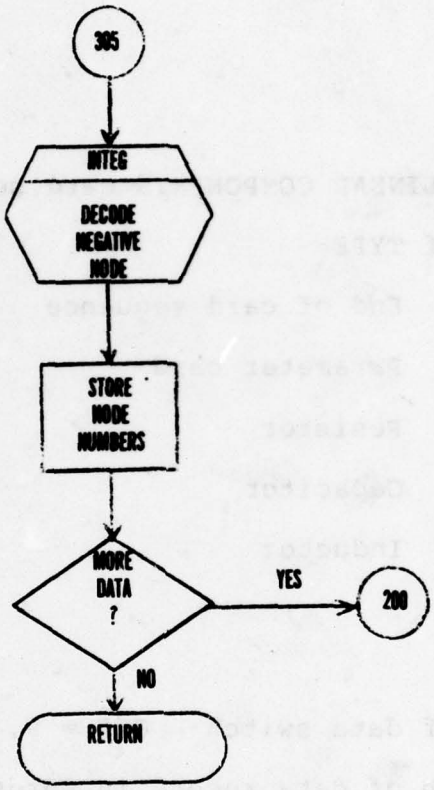
Subroutine MODIN initiates the processing of a MODIFY card sequence. Rather than processing the input cards itself, it resets MODE to identify the device being modified, turns the modify switch on (MODFY=1), and returns to the calling program where the remaining cards of the sequence are processed according to the MODE of the device being modified.

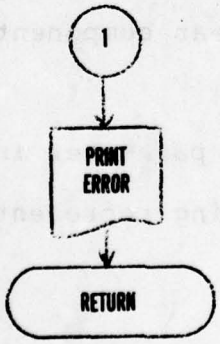

NCIN
ALPHA
DATAWR
DECIM
ERROR
INTEG
DRVOUT
SHIFT
4
2
2
SCAN
SCAN
SCAN
DRIVWR
RSHIFT
PACKR
PACKL
PACKR
PACKL
PACKR
PACKL


START
TEST TYPE
100 1
200 2
800 8
900 9
1000 10
OTHER 1
100
101
ANY COMPONENTS INPUT ?
NO
PRINT ERROR
RETURN
YES
102
INPUT ERRORS ?
YES
NO
104
DATAWR
WRITE DATA RECORD TO FILE 20
STORE DATA RECORD NUMBER
DRVOUT
WRITE DRIVER RECORD TO FILE 21
103
RETURN


200
INCREMENT WORD COUNT
DECIM
DECODE NEXT COEFFICIENT
MORE DATA ?
NO
203
SAVE WORD COUNT
YES
202
LAST COEFFICIENT ?
NO
RETURN
YES
RETURN


800
900
1000
RESET WORD COUNT
INCREMENT DATA RECORD LENGTH
CALCULATE NUMBER OF COMPONENTS
301
TOO MANY COMPONENTS ?
YES
PRINT ERROR
RETURN
NO
302
STORE COMPONENT TYPE
ALPHA
DECODE "R","C","L"
303
MORE DATA ?
NO
PRINT ERROR
RETURN
YES
304
INTEG
DECODE POSITIVE MODE
MORE DATA ?
NO
YES
305


305
INTEG
DECODE
NEGATIVE
MODE
STORE
MODE
NUMBERS
MORE
DATA
?
YES
200
NO
RETURN


1
PRINT
ERROR
RETURN

NAME: NCIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *NONLINEAR COMPONENTS card sequence according to the value of TYPE:

Type = 1 End of card sequence

Type = 2 Parameter card

Type = 8 Resistor

Type = 9 Capacitor

Type = 10 Inductor

VARIABLES:

ISW = End of data switch. Off = 0, On = 1

NCOMP = Length of data record in words

NODEA = Positive node of connection

NODEB = Negative node of connection

NUM = Number of nonlinear components this card sequence

NW = Word counter for parameter input

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

ALPHA DATAWR DECIM DRVOUT ERROR INTEG

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine NCIN is the input processor for NONLINEAR COMPONENTS card sequences. Its function is to decode the nonlinear components input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine NCIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine NCIN which processes the card according to its TYPE as follows:

TYPE = 1 End of Card Sequence

The closing of a nonlinear components definition begins by testing the input for errors. If no linear component cards were processed (NCOMP.LE.0) or if the input error switch is on (JABT=1), return is made to the calling program with an appropriate error condition. Otherwise the data record is written to file 20 from the first NCOMP words of the common data buffer BUFF by subroutine DATAWR.

The nonlinear components data record is structured as a series of NUM contiguous 13-word entries, one for each nonlinear

component defined in the card sequence:

BUFF(1)-(13)	First Component Definition
BUFF(14)-(26)	Second Component Definition
.	.
.	.
.	.
BUFF(NCOMP-12)-(NCOMP)	Last Component Definition

The first word of each entry contains an integer code representing the component type: 1=resistor, 2=capacitor, 3=inductor. The second and third words represent the positive and negative nodes of connection. The last ten words contain from one to ten nonlinear coefficients which define the element:

BUFF(1),(14)...(NCOMP-12)	Component Codes
BUFF(2),(15)...(NCOMP-11)	Positive Nodes of Connection
BUFF(3),(16)...(NCOMP-10)	Negative Nodes of Connection
BUFF(4),(17)...(NCOMP-9)	First Order Coefficients
BUFF(5),(18)...(NCOMP-8)	Second Order Coefficients
. . .	.
. . .	.
. . .	.
BUFF(13),(26)...(NCOMP)	Tenth Order Coefficients

The nonlinear components driver record is completed and written by subroutine DRVOUT to file 21 from the first ten words of global common, where·

MODE=5	Nonlinear Component Identifier
STADD	Data File Record Number

LNGTH=NCOMP Length of Data Record in Words

MISC(1)=NCOMP Number of Nonlinear Components x 4

Other MISC driver parameters are not used.

TYPE=2 Parameter Card

From one to ten decimal parameters are extracted from the card image, formatted, and stored in the data buffer by subroutine DECIM. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of coefficient values input on previous data cards. Parameter storage starts at BUFF(NCOMP-10+NW) where NW=NXT+1 and proceeds through successive buffer locations as coefficient values are decoded by subroutine DECIM. If an end-of-data is encountered (ISW=1) before BUFF(NCOMP) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation of additional parameter cards.

TYPE=8, 9, OR 10 Resistor, Capacitor, or Inductor Card

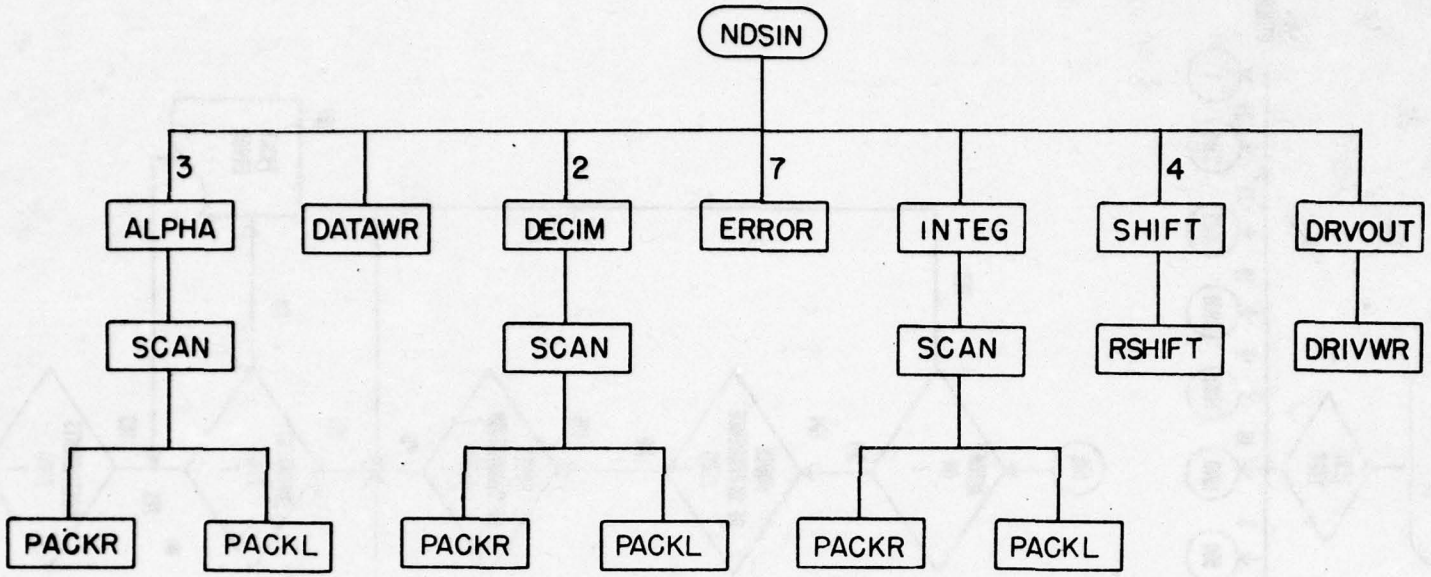
The parameter count is reset to NXT=0, the data record length is incremented to accommodate a new 13-word component definition, and the component count NUM is calculated. If more

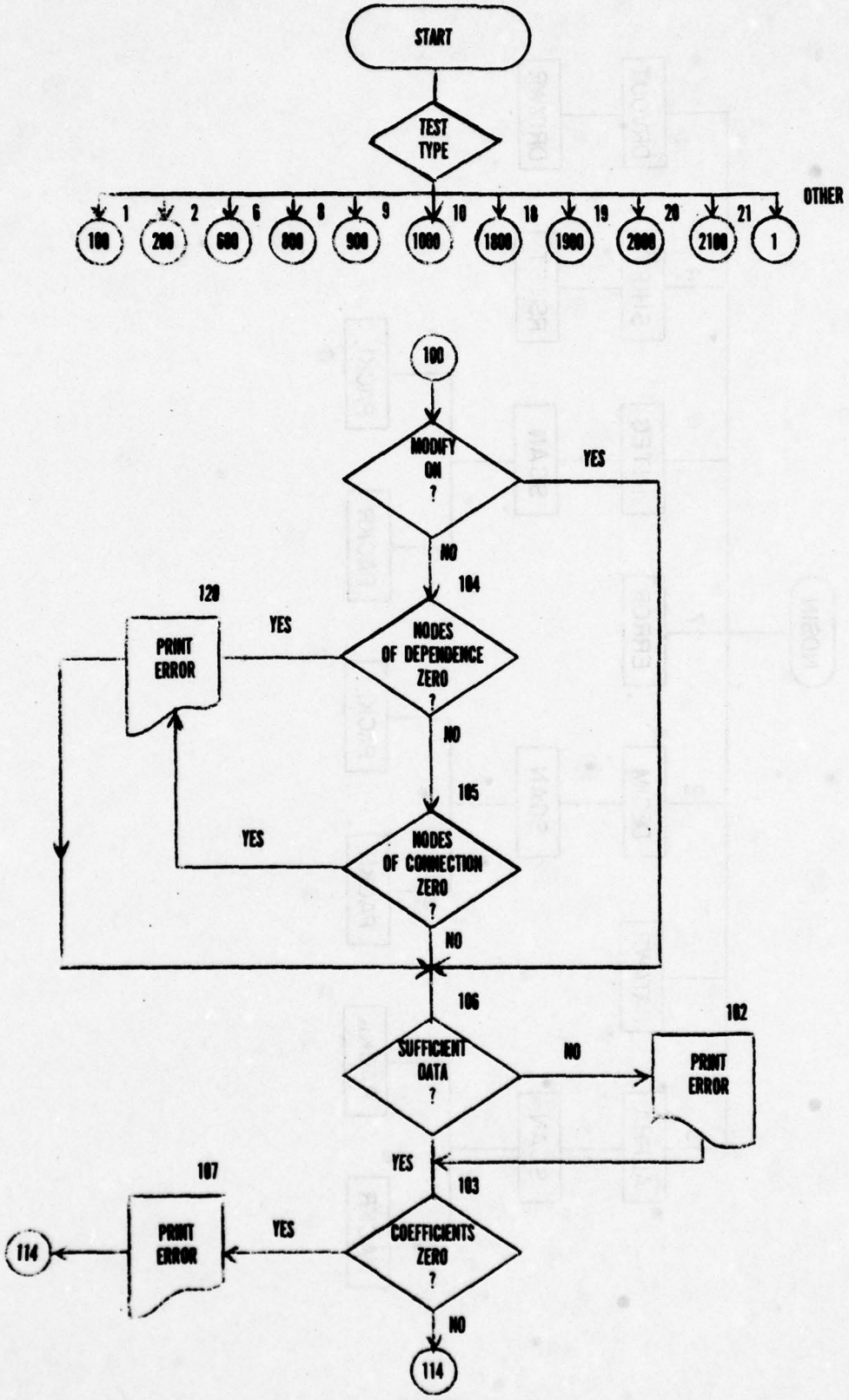
than 10 nonlinear components have been processed in the card sequence (NUM.GT.10), subroutine NCIN exits with an appropriate error condition. Otherwise the new component code is calculated by TYPE-7 and stored in BUFF(NCOMP-12).

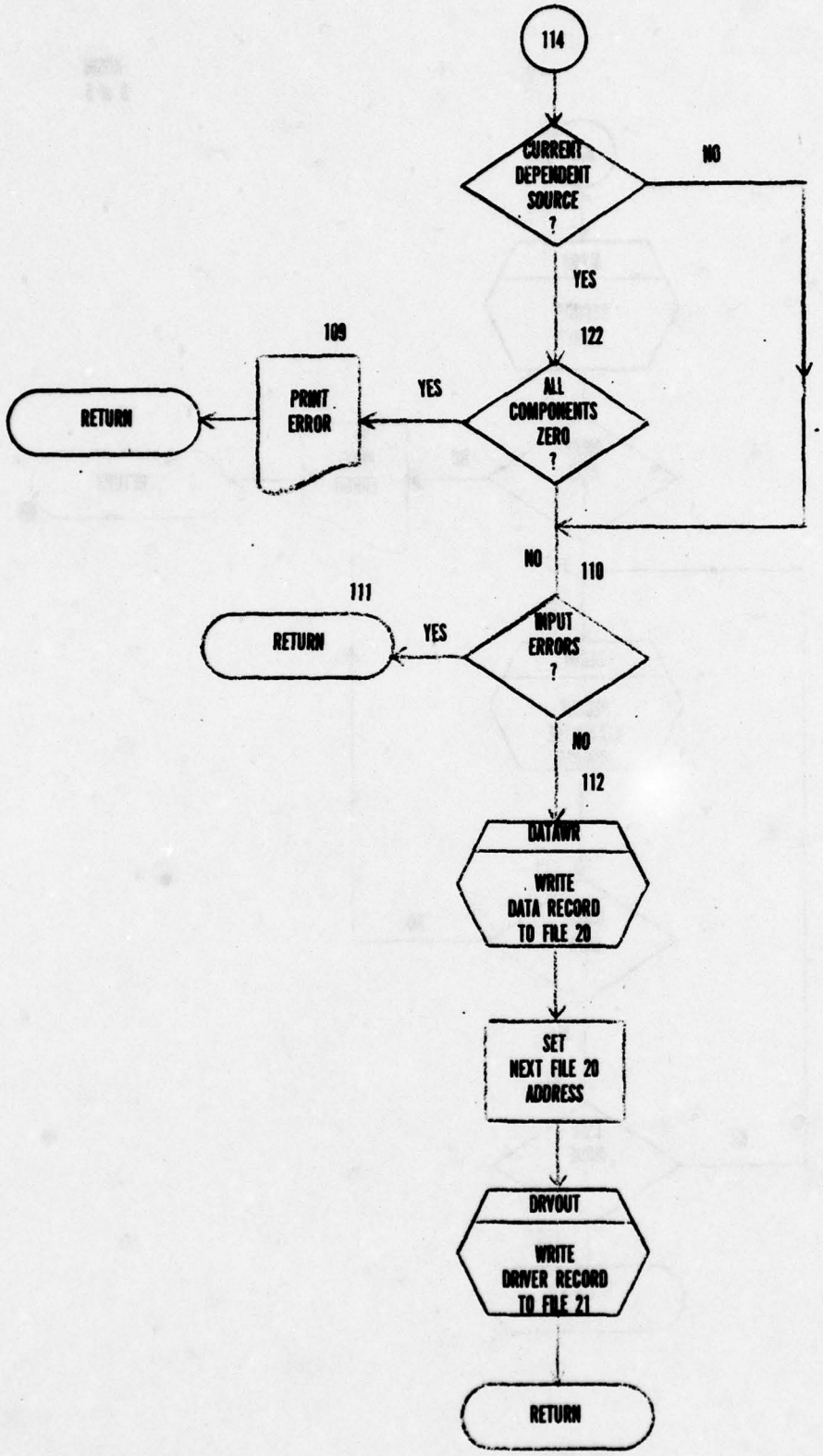
For each nonlinear component card, subroutine NCIN anticipates the input of a component identifier and two nodes of connection on one card. During decoding of the input card, if an end-of-data is encountered (ISW=1) before these parameters have been processed, subroutine NCIN exits with an appropriate error condition.

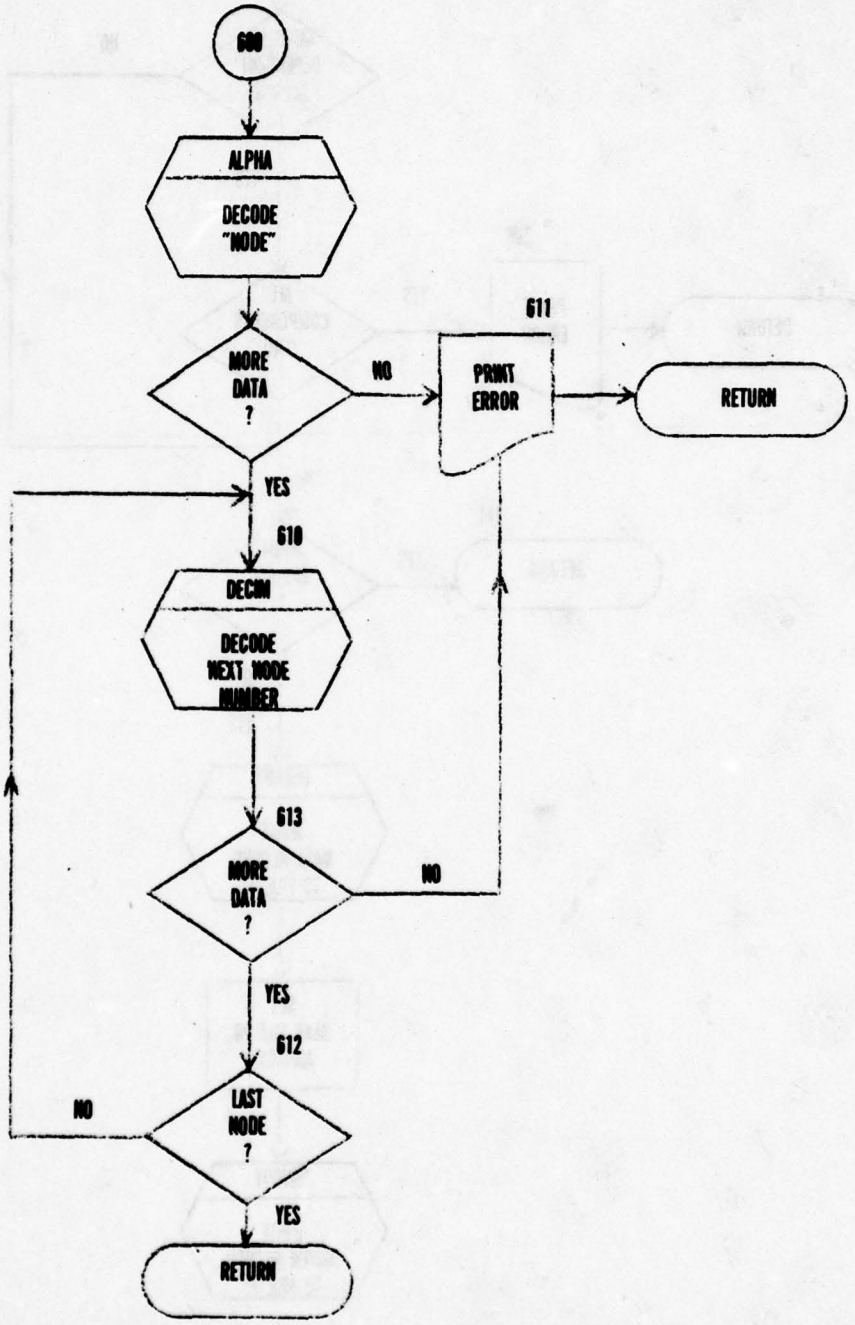
The R, C, or L identifier is decoded by subroutine ALPHA. The nodes of connection are decoded and formatted by subroutine DECIM and stored in BUFF(NCOMP-11) and BUFF(NCOMP-10). If the end-of-data switch is off (ISW=0), control transfers to decode the coefficients which follow on the card according to TYPE=2 processing. If the end-of-data switch is on (ISW=1), subroutine NCIN returns to the calling program.

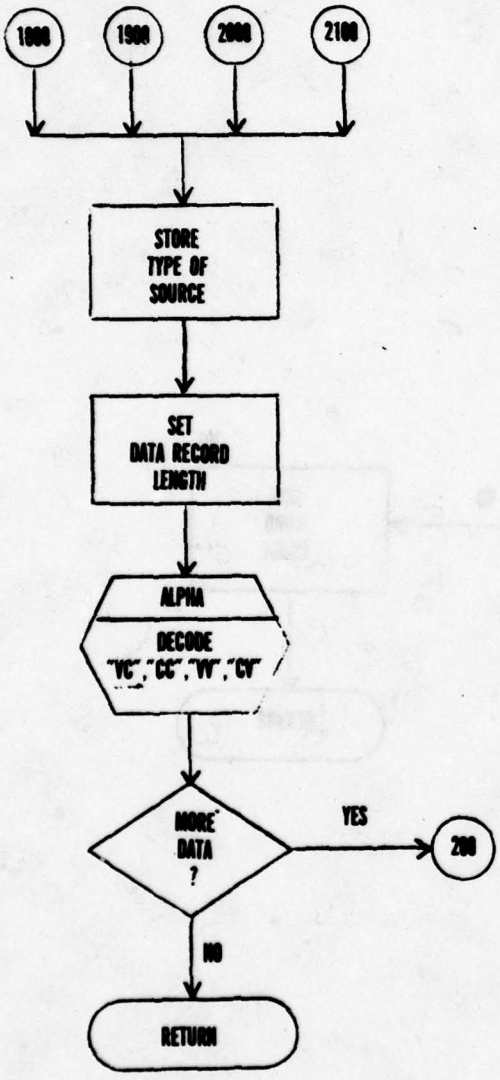
Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.

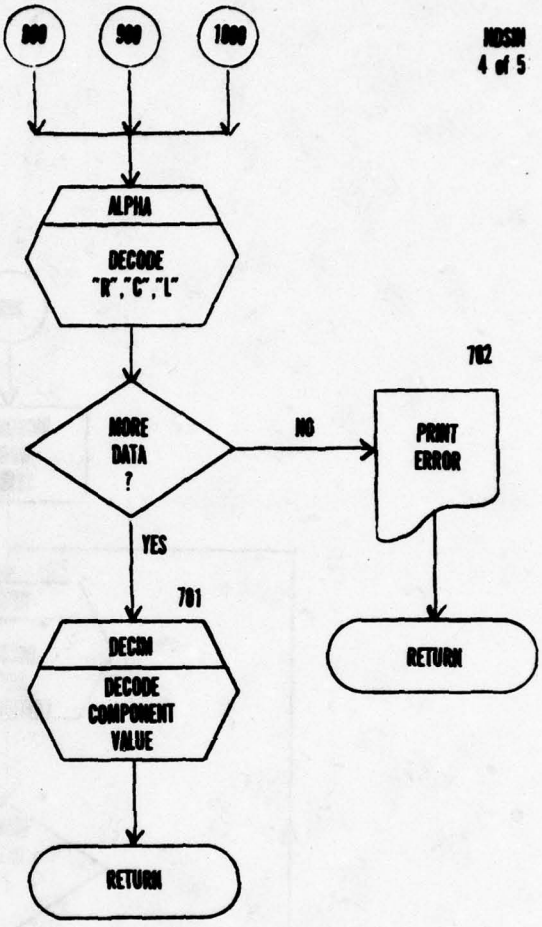

NDSIN
3
2
7
4
ALPHA
DATAWR
DECIM
ERROR
INTEG
SHIFT
DRVOUT
SCAN
SCAN
SCAN
RSHIFT
DRIVWR
PACKR
PACKL
PACKR
PACKL
PACKR
PACKL

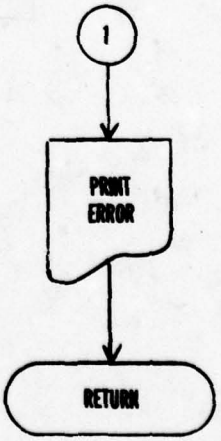

START
TEST TYPE
OTHER
1
2
6
8
9
10
18
19
20
21
100
200
600
800
900
1000
1800
1900
2000
2100
1
100
MODIFY ON ?
YES
NO
104
NODES OF DEPENDENCE ZERO ?
YES
120
PRINT ERROR
NO
105
NODES OF CONNECTION ZERO ?
YES
NO
106
SUFFICIENT DATA ?
NO
102
PRINT ERROR
YES
103
COEFFICIENTS ZERO ?
YES
107
PRINT ERROR
114
NO
114

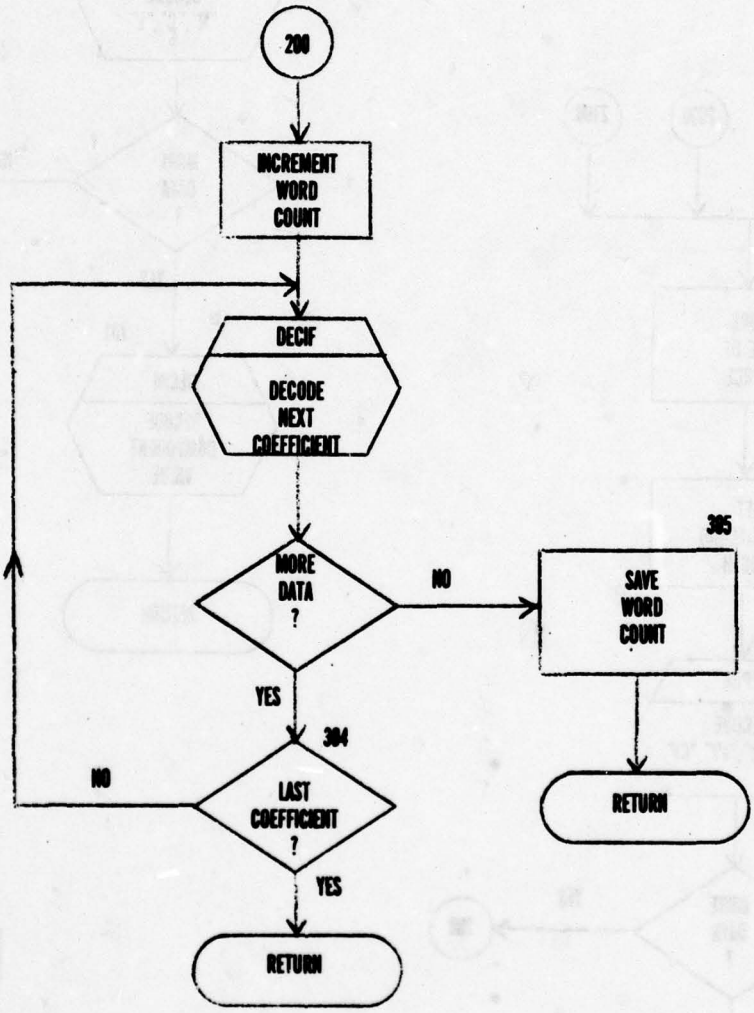

114
CURRENT DEPENDENT SOURCE ?
NO
YES
122
ALL COMPONENTS ZERO ?
YES
109
PRINT ERROR
RETURN
NO
110
INPUT ERRORS ?
YES
111
RETURN
NO
112
DATAWR
WRITE DATA RECORD TO FILE 20
SET NEXT FILE 20 ADDRESS
DRVOUT
WRITE DRIVER RECORD TO FILE 21
RETURN


600
ALPHA
DECODE "NODE"
611
MORE DATA ?
NO
PRINT ERROR
RETURN
YES
610
DECIM
DECODE NEXT NODE NUMBER
613
MORE DATA ?
NO
YES
612
LAST NODE ?
NO
YES
RETURN


1000
1900
2000
2100
STORE TYPE OF SOURCE
SET DATA RECORD LENGTH
ALPHA
DECODE "VC","CC","VV","CV"
MORE DATA ?
YES
200
NO
RETURN


800
900
1000
ALPHA
DECODE "R","C","L"
MORE DATA ?
NO
702
PRINT ERROR
RETURN
YES
701
DECIM
DECODE COMPONENT VALUE
RETURN


1
PRINT ERROR
RETURN


200
INCREMENT WORD COUNT
DECIF
DECODE NEXT COEFFICIENT
MORE DATA ?
NO
305
SAVE WORD COUNT
YES
304
LAST COEFFICIENT ?
NO
RETURN
YES
RETURN

NAME: NDSIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *LINEAR DEPENDENT SOURCE card sequence according to the value of TYPE:

Type = 1	End of card sequence
Type = 2	Parameter card
Type = 6	Node card
Type = 8	Resistor
Type = 9	Capacitor
Type = 10	Inductor
Type = 18	VC
Type = 19	CC
Type = 20	VV
Type = 21	CV

VARIABLES:

A	= Positive node of connection
B	= Negative node of connection
C	= Value of parallel capacitor
ISW	= End of data switch: off = 0, on = 1
L	= Value of parallel inductor
NW	= Word counter for parameter input
R	= Value of parallel resistor
STRING	= Alphanumeric string representation of input

value

SOURCE = Type of source; 1 = VC

2 = CC

3 = VV

4 = CV

XX = Positive node of dependence

YY = Negative node of dependence

SUBROUTINES CALLED:

ALPHA DATAWR DECIM DRIVWR

ERROR INTEG

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine NDSIN is the input processor for NONLINEAR DEPENDENT SOURCE card sequences. Its function is to decode the nonlinear source input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine NDSIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine NDSIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

The closing of a nonlinear dependent source definition begins by testing the input for errors. Subroutine NDSIN exits with an appropriate error message if any of the following input errors are detected:

1) All four nodes (A, B, XX, and YY) are zero and the card sequence is not a modify function (MODFY=0)
2) No coefficients were processed (NXT=0)
3) The coefficients are all zero
4) For current controlled sources, all three parallel components (R, C, and L) are zero
5) The input error switch is on (JABT=1)

Otherwise the nonlinear dependent source data record is written to file 20 from the common data buffer BUFF and the file 20 record number DATREC is updated to point beyond the new data record.

The data record allocates sufficient disk storage for the nonlinear dependent source input values as well as all other data generated internally by its model in subsequent NCAP phases. The length of a nonlinear dependent source data record depends on the type of source specified: voltage controlled sources require 20 words of storage for input values only, while current controlled sources require 86 words of storage as follows:

BUFF(1)-(20)	Input Values
BUFF(21)-(23)	Component Values
BUFF(24)-(86)	Impedance Table

After the data record has been written to disk, subroutine DRVOUT is called to complete the driver record and write it to file 21. The nonlinear dependent source driver record, taken from the first ten words of global common, consists of:

MODE=26	Nonlinear Dependent Source Identifier
STADD	Data File Record Number
LNGTH=20 or 86	Length of Data Record in Words
MISC(1)=SOURCE	Source Code: 1=VC, 2=CC, 3=VV, 4=CV
MISC(3)=XX	Positive Node of Dependence
MISC(4)=YY	Negative Node of Dependence
MISC(5)=A	Positive Node of Connection
MISC(6)=P	Negative Node of Connection

After returning from subroutine DRVOUT, subroutine NDSIN exits to the calling program.

TYPE=2 Parameter Card

From one to ten nonlinear coefficients are extracted from the card image, formatted, and stored in the data buffer. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of

coefficients input on previous data cards. Coefficient storage begins at BUFF(NW*2-1) where NW=NXT+1, and proceeds through successive odd buffer locations as the coefficients are decoded by subroutine DECIM. The corresponding even buffer locations BUFF(NW*2), which are allocated for future implementation of imaginary parts of complex coefficients, are zeroed. If an end-of-data is encountered (ISW=1) before BUFF(19) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation of additional parameter cards.

TYPE=6 Node Card

The NODE identifier is decoded by subroutine ALPHA. Then the nodes of dependence and nodes of connection are extracted from the card image and formatted by subroutine INTEG and stored in MISC(3) through MISC(6). If an end-of-data is encountered (ISW=1) before four node numbers have been decoded, subroutine NDSIN exits with an appropriate error condition.

TYPE=8, 9, or 10 Resistor, Capacitor, or Inductor

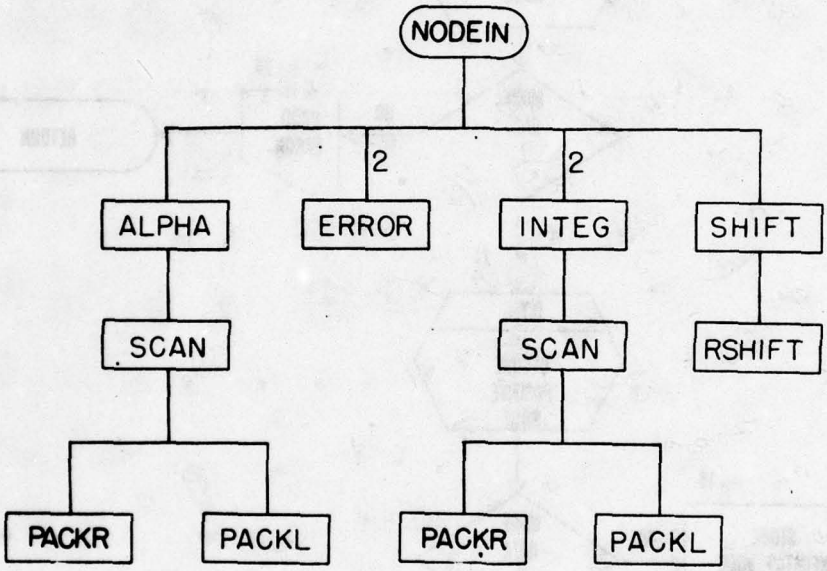
The R, C, or L identifier is decoded by subroutine ALPHA. If an end-of-data is encountered (ISW=1), subroutine NDSIN exits with an appropriate error condition. Otherwise the component value is extracted from the card image, formatted, and stored at

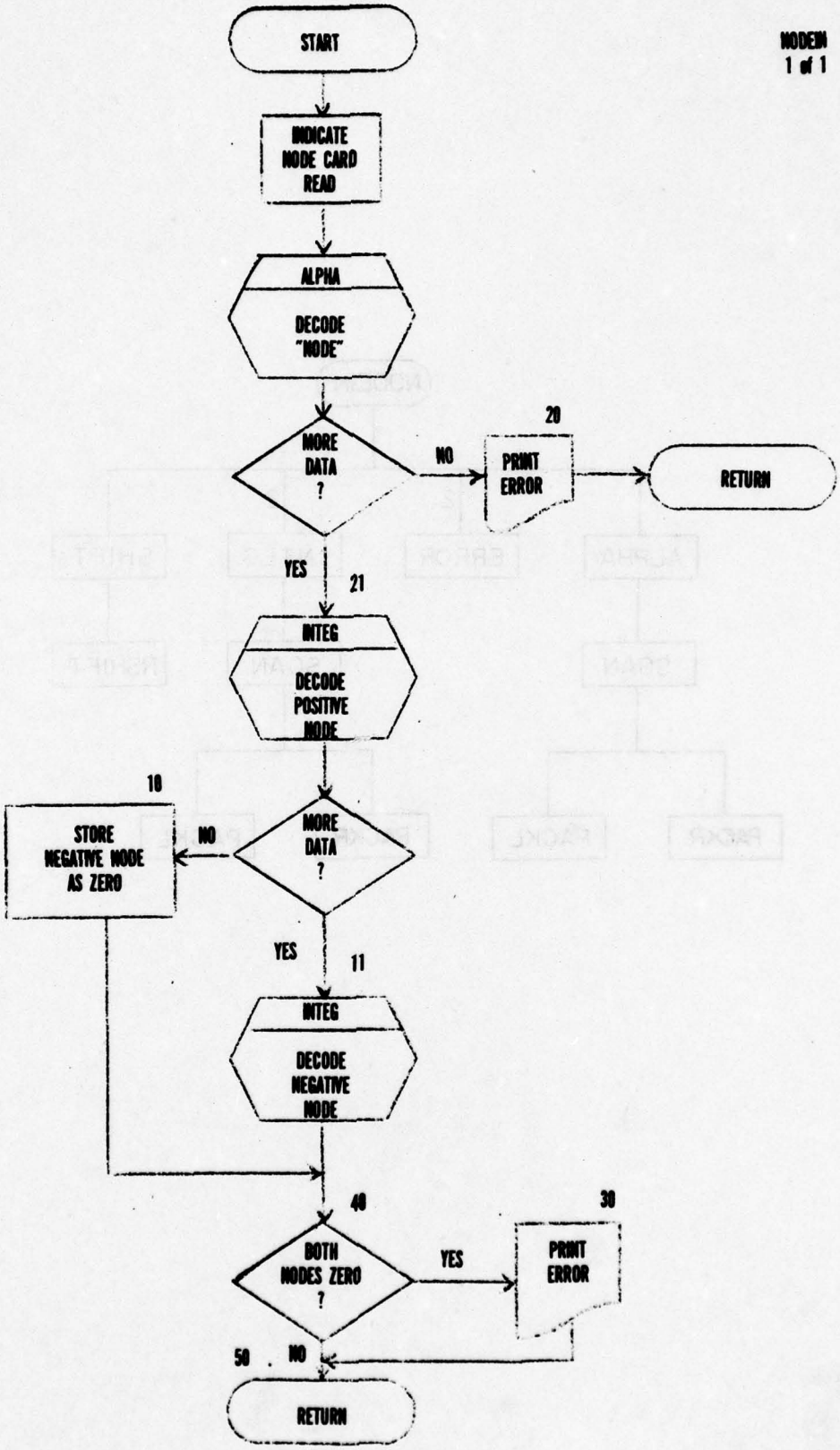
BUFF(TYPE+13) by subroutine DECIM.

TYPE= 18, 19, 20, or 21 VC, CC, VV, or CV Card

The SOURCE type is calculated according to TYPE-17 and the data record length is set: LNGTH=20 for voltage controlled sources, LNGTH=86 for current controlled sources. The VC, CC, VV, or CV identifier is decoded by subroutine ALPHA. If the end-of-data is encountered (ISW=1), subroutine NDSIN returns to the calling program. Otherwise control transfers to TYPE=2 processing for the remaining data on the card.

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


NODEIN
2
2
ALPHA
ERROR
INTEG
SHIFT
SCAN
SCAN
RSHIFT
PACKR
PACKL
PACKR
PACKL


START
INDICATE MODE CARD READ
ALPHA
DECODE "MODE"
MORE DATA ?
NO
20
PRINT ERROR
RETURN
YES
21
INTEG
DECODE POSITIVE MODE
10
STORE NEGATIVE MODE AS ZERO
NO
MORE DATA ?
YES
11
INTEG
DECODE NEGATIVE MODE
40
BOTH MODES ZERO ?
YES
30
PRINT ERROR
50
NO
RETURN

NAME: NODEIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes NODE cards

VARIABLES:

ISW = End of data switch: Off = 0, On = 1

NODEA = Positive node of connection

NODEB = Negative node of connection

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

ALPHA ERROR INTEG

CALLING PROGRAMS:

GENIN JFETIN PTIN SDIN TRNIN

VDIN VTIN

DESCRIPTION:

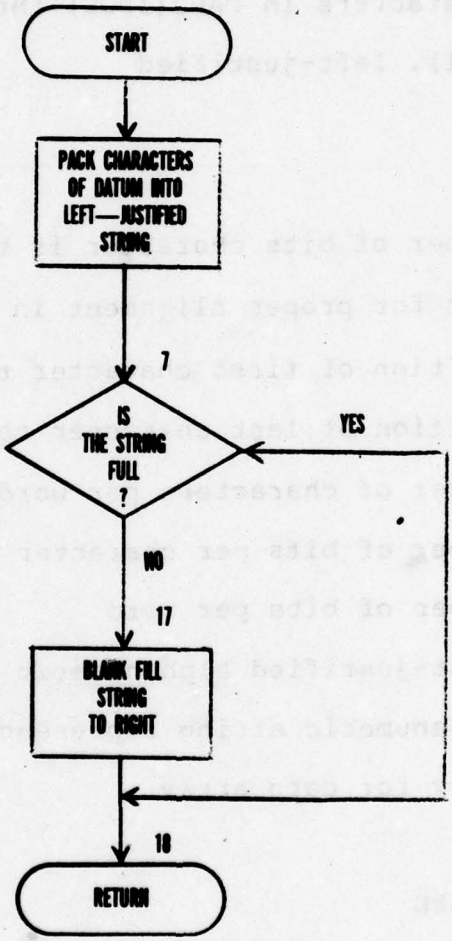
Subroutine NODEIN decodes and formats the standard NCAP node card. The card image is transmitted through common storage to subroutine NODEIN where it is processed according to an anticipated format. The node specifications are returned to the

calling program through the arguments NODEA and NODEB.

First the node error switch is turned off (NABT=Ø) and the NODE identifier is decoded by subroutine ALPHA. If an end-of-data is encountered (ISW=1), subroutine NODEIN exits with an appropriate error condition.

If the end-of-data switch is off (ISW=Ø), the positive node of connection is extracted from the card image, formatted, and stored in NODEA by subroutine INTEG. If the end-of-data is encountered (ISW=1), the negative node of connection is assumed to be ground (NODEB=Ø). Otherwise the negative node is extracted from the card image, formatted and stored in NODEB by subroutine INTEG.

In the event that both nodes are zero, subroutine NODEIN exits with an appropriate error condition.


START
PACK CHARACTERS OF DATUM INTO LEFT—JUSTIFIED STRING
7
IS THE STRING FULL ?
YES
NO
17
BLANK FILL STRING TO RIGHT
18
RETURN

NAME: PACKL

TYPE: SUBROUTINE

GENERAL PURPOSE:

Packs the characters in CARD(IPOS) through CARD(J-1) into STRING(1), left-justified

VARIABLES:

BIT	= Number of bits character is to be shifted left for proper alignment in packed notation
IPOS	= Position of first character to be packed
J	= Position of last character to be packed + 1
NCHAR	= Number of characters per word
NBCHAR	= Number of bits per character
NBWORD	= Number of bits per word
RBLANK	= Right-justified alphanumeric blank
STRING	= Alphanumeric string representation returned
STOP	= Index for card array

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

SCAN

DESCRIPTION:

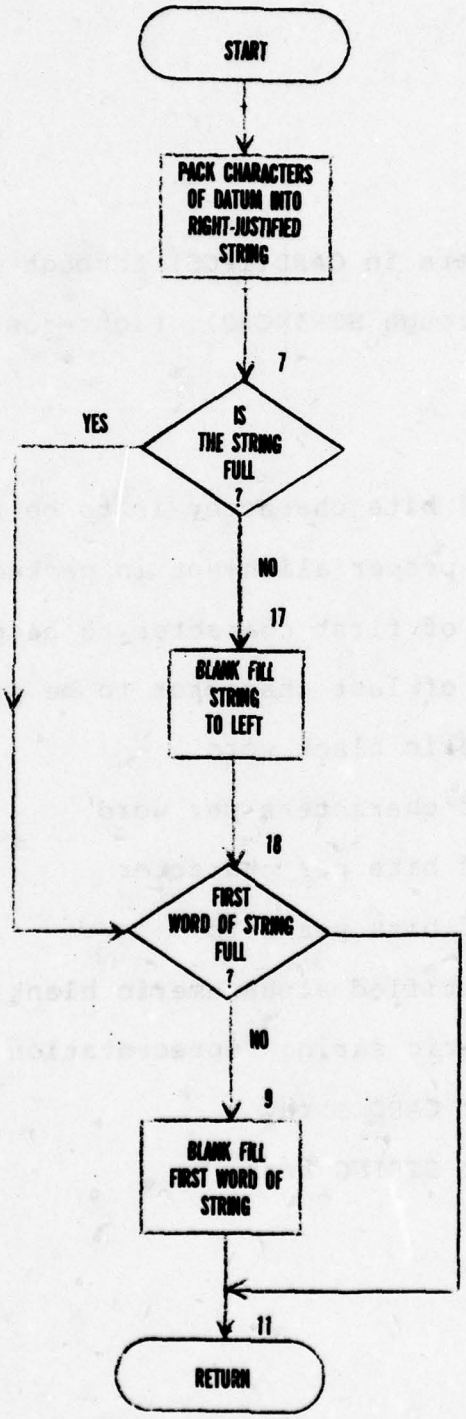
Subroutine PACKL belongs to a group of Phase 0 character manipulation routines which format NCAP input data. Its function is to pack the characters comprising an alphabetic datum extracted by subroutine SCAN into a left-justified alphanumeric string in preparation for formatting by subroutine ALPHA.

Input to the subroutine is the right-justified card image stored in the integer array CARD, the character position IPOS at which the datum begins, and the character position J at which the datum ends. Output from the routine is the left-justified alphanumeric string representation returned in STRING(1). The required machine dependent character manipulation parameters NCHAR (number of characters per word), NBCHAR (number of bits per character), and NBWORD (number of bits per word) are transmitted to subroutine PACKL through the labelled common area WORDSZ.

The NBWORD bits of STRING(1) contain NCHAR characters, each occupying NBCHAR bits. The characters are stored in the string from left to right, where the leftmost character of the datum taken from CARD (IPOS) is placed in the most significant bits of STRING(1) and so on.

The packing technique is arithmetic: each right-justified character to be packed, CARD(STOP), is shifted left into the correct position by multiplication by an appropriate power of 2, (2 ** BIT), and then simply added to the string, STRING(1). The variable STOP indexes the CARD array, while BIT represents the number of bits a character must be shifted left for proper positioning within the string.

If fewer than NCHAR characters are input to the routine, the string is filled to the right with alphanumeric blanks. If more than NCHAR characters are input, the right-most characters are truncated.


START
PACK CHARACTERS OF DATUM INTO RIGHT-JUSTIFIED STRING
7
IS THE STRING FULL ?
YES
NO
17
BLANK FILL STRING TO LEFT
18
FIRST WORD OF STRING FULL ?
NO
9
BLANK FILL FIRST WORD OF STRING
11
RETURN

NAME: PACKR

TYPE: SUBROUTINE

GENERAL PURPOSE:

Packs the characters in CARD(IPOS) through CARD(J-I) into STRING(1) through STRING(2), right-justified

VARIABLES:

BIT = Number of bits character is to be shifted left for proper alignment in packed notation

IPOS = Position of first character to be packed

J = Position of last character to be packed + 1

LBLANK = Alphanumeric blank word

NCHAR = Number of characters per word

NBCHAR = Number of bits per character

NBWORD = Number of bits per word

RBLANK = Right-justified alphanumeric blank

STRING = Alphanumeric string representation returned

STOP = Index for CARD array

WORD = Index for STRING array

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

SCAN

DESCRIPTION:

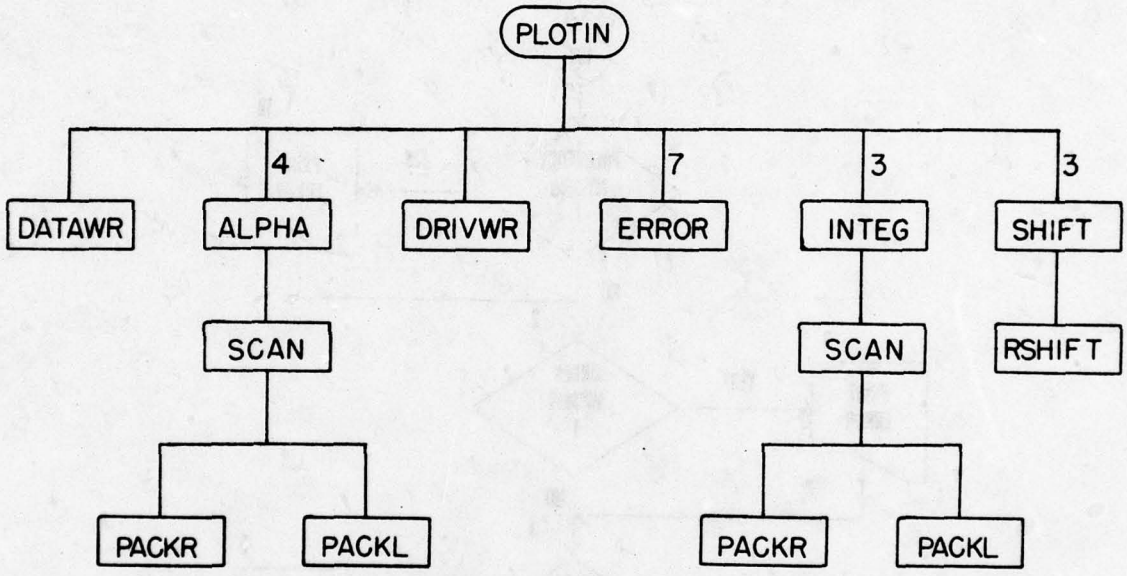
Subroutine PACKR belongs to a group of Phase 0 character manipulation routines which format NCAP input data. Its function is to pack the characters comprising a numeric data value extracted by subroutine SCAN into a right-justified alphanumeric string in preparation for formatting by subroutines DECIM and INTEG.

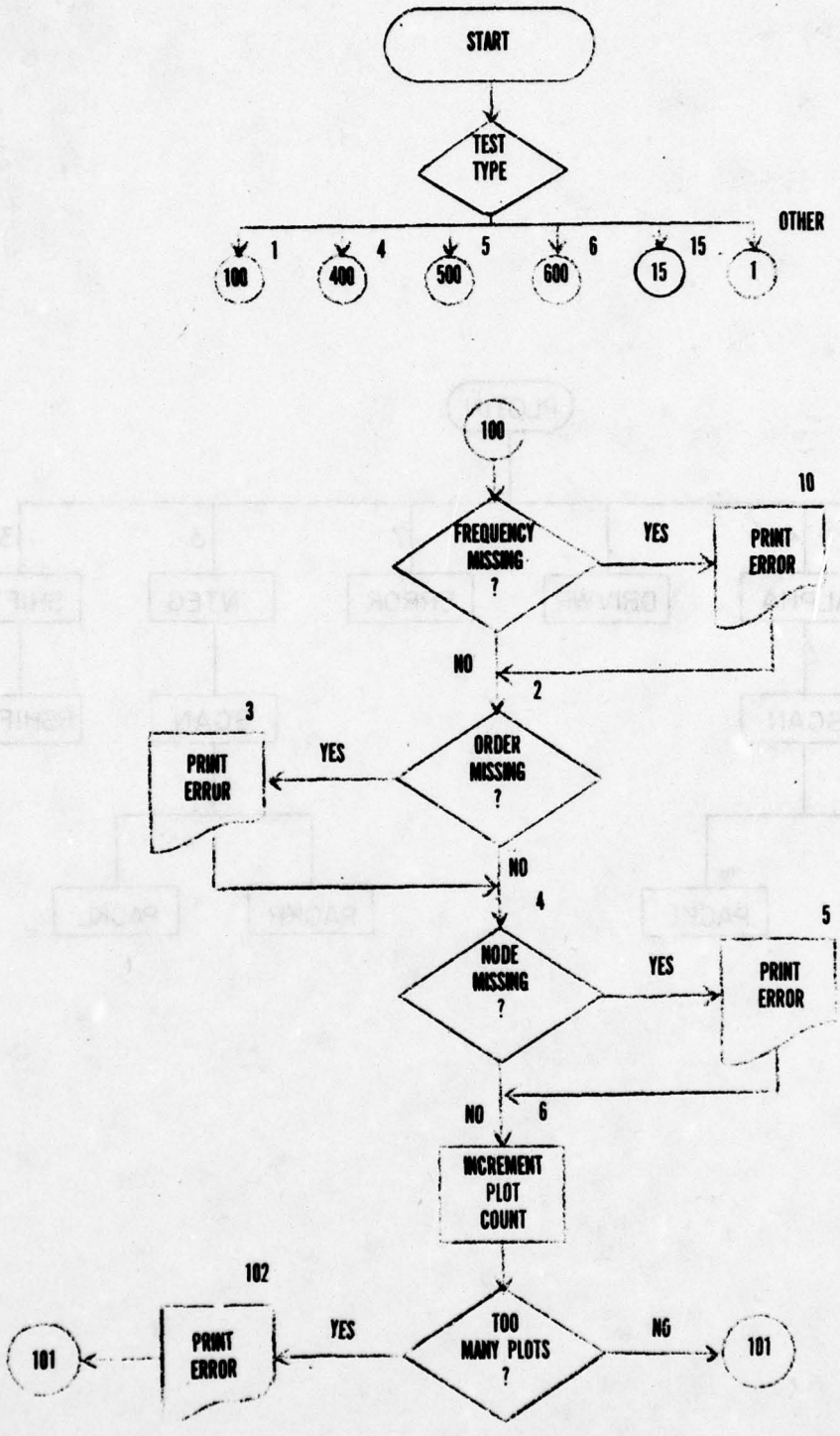
Input to the routine is the right-justified card image stored in the integer array CARD, the character position IPOS at which the datum begins, and the character position J at which the datum ends. Output from the routine is the right-justified alphanumeric string representation of the datum returned in STRING(1) and STRING(2). The required machine dependent character manipulation parameters NCHAR (number of characters per word), NBCHAR (number of bits per character), and NBWORD (number of bits per word) are transmitted to subroutine PACKR through the labelled common area WORDSZ.

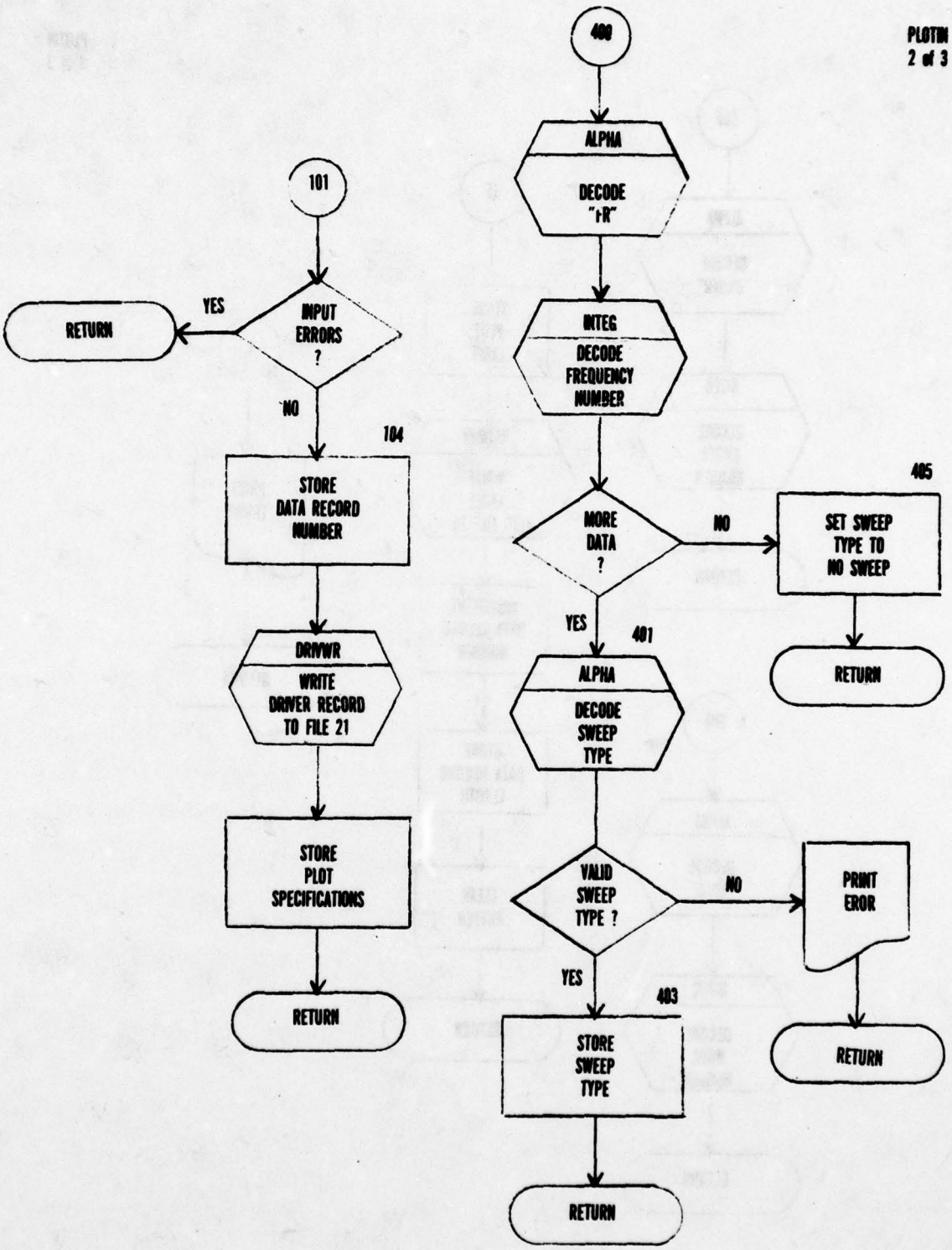
The NBWORD bits of each word of the alphanumeric string representation contain NCHAR characters, each occupying NBCHAR bits. The characters are stored from right to left (from the least significant to the most significant bit), where the least significant character of the string occupies bit positions 1 through NBCHAR and so on. STRING(2) is packed with the NCHAR characters from CARD(J-1), CARD(J-2)..., CARD(J-NCHAR). STRING(1) is packed with the remaining characters through CARD(IPOS). The elements of STRING are blank filled to the left if necessary.

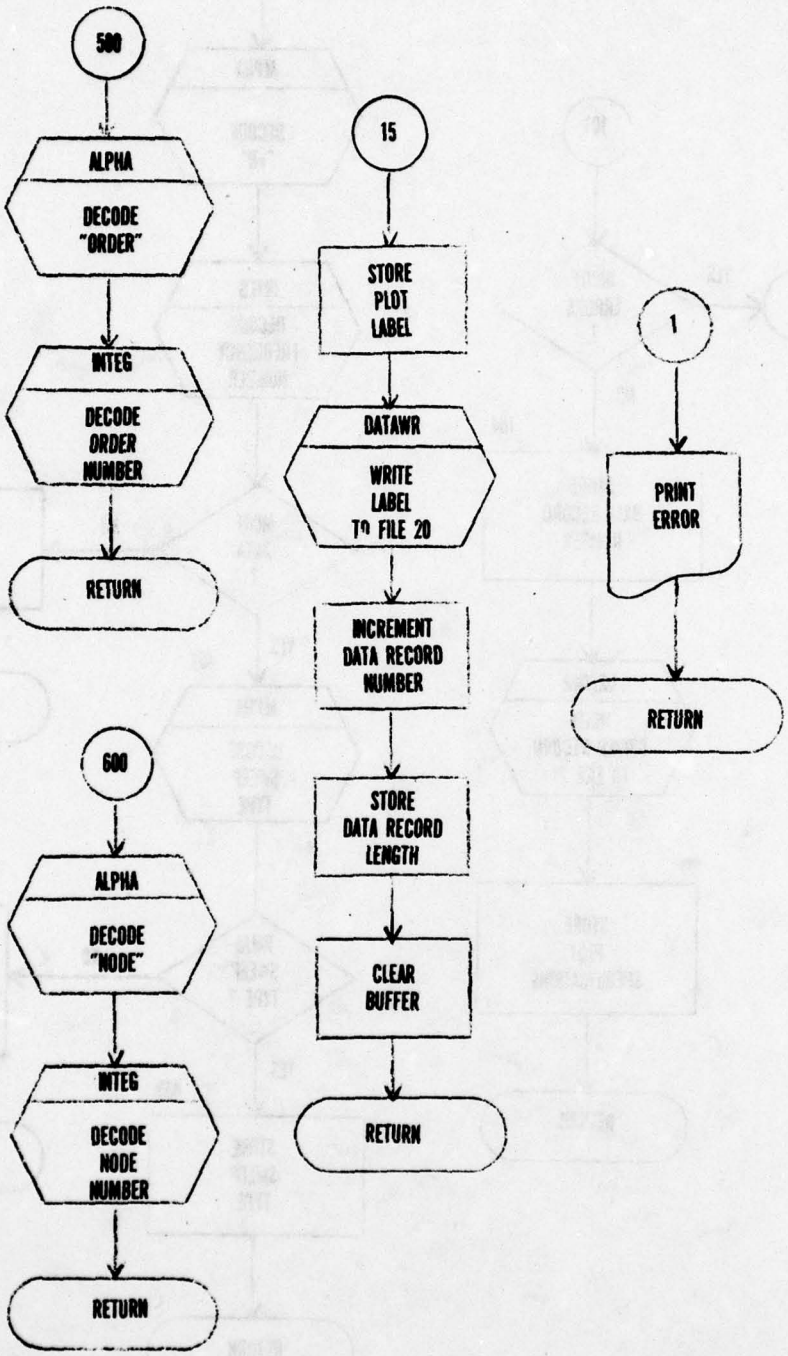
The packing technique is arithmetic: each right-justified character to be packed, CARD(STOP), is shifted left into the correct position by multiplication by an appropriate power of 2, (2 * * BIT), and then simply added to the string at STRING(WORD). The variables STOP and WORD index the CARD and STRING arrays, while BIT represents the number of bits a character must be shifted left for proper positioning within the string.

If fewer than 2 x NCHAR characters are input to the routine, the string is filled to the left with alphanumeric blanks. If more than 2 x NCHAR characters are input, the more significant characters are truncated.


PLOTIN
4
7
3
3
DATAWR
ALPHA
DRIVWR
ERROR
INTEG
SHIFT
SCAN
SCAN
RSHIFT
PACKR
PACKL
PACKR
PACKL


START
TEST TYPE
OTHER
1
100
4
400
5
500
6
600
15
15
1
100
10
FREQUENCY MISSING ?
YES
PRINT ERROR
NO
2
3
PRINT ERROR
YES
ORDER MISSING ?
NO
4
5
NODE MISSING ?
YES
PRINT ERROR
NO
6
INCREMENT PLOT COUNT
102
101
PRINT ERROR
YES
TOO MANY PLOTS ?
NO
101


400
ALPHA
DECODE
"FR"
INTEG
DECODE
FREQUENCY
NUMBER
MORE
DATA
?
NO
405
SET SWEEP
TYPE TO
NO SWEEP
RETURN
YES
401
ALPHA
DECODE
SWEEP
TYPE
VALID
SWEEP
TYPE ?
NO
PRINT
EROR
RETURN
YES
403
STORE
SWEEP
TYPE
RETURN
101
INPUT
ERRORS
?
YES
RETURN
NO
104
STORE
DATA RECORD
NUMBER
DRIVWR
WRITE
DRIVER RECORD
TO FILE 21
STORE
PLOT
SPECIFICATIONS
RETURN


500
ALPHA
DECODE "ORDER"
INTEG
DECODE ORDER NUMBER
RETURN
600
ALPHA
DECODE "NODE"
INTEG
DECODE NODE NUMBER
RETURN
15
STORE PLOT LABEL
DATAWR
WRITE LABEL TO FILE 20
INCREMENT DATA RECORD NUMBER
STORE DATA RECORD LENGTH
CLEAR BUFFER
RETURN
1
PRINT ERROR
RETURN

NAME: PLOTIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *PLOT card sequence according to the value of TYPE

Type = 1 End of card sequence

Type = 4 Frequency

Type = 5 Order

Type = 6 Node

Type = 15 Label

VARIABLES:

IFR = User-assigned number of frequency to be plotted

INOD = Nodes to be plotted

IORD = Orders to be plotted

ISW = End of data switch. Off = Ø, On = 1

ITYP = Type of plot: 1 = Linear
2 = Logarithmic

STRING = Alphanumeric string representation of input value

TYPES = Valid frequency sweep types

SUBROUTINES CALLED:

ALPHA DATAWR DRIVWR ERROR INTEG

CALLING PROGRAMS:

PRCSS

DESCRIPTION·

Subroutine PLOTIN is the input processor for PLOT card sequences. Its function is to decode the plot input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine PLOTIN causes a single NCAP input card to be processed. The card image and its TYPE are transmitted through common storage to subroutine PLOTIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

The closing of a plot definition begins by testing the input for errors. Subroutine PLOTIN exits with an appropriate error message if any of the following input errors are detected:

1) No frequency data was processed (MISC(1)=Ø)
2) No order data was processed (MISC(3)=Ø)
3) No node data was processed (MISC(4)=Ø)
4) More than 1Ø plot specifications were processed (NPLOT.GT.1Ø)

5) The input error switch is on (JABT=1)

If no input errors are detected, the file 20 record number DATREC is updated to point beyond the plot data record and subroutine DRIVWR is called to write the driver record to file 21.

The PLOT driver record, taken from the first ten words of global common, consists of:

MODE=22	Plot Identifier
STADD	Data File Record Number
LNGTH=0,50, or 100	Length of Data Record in Words
MISC(1)	Frequency Number
MISC(2)	Type of Plot: 1=Linear
	2-Logarithmic
MISC(3)	Order Number
MISC(4)	Node Number

Other MISC driver parameters are not used.

After the driver record has been written to disk, the plot specifications are appended to the plot table in the labelled common area PLOTFL and subroutine PLOTIN returns to the calling program.

TYPE=4 Frequency Card

The FR identifier is decoded by subroutine ALPHA and the frequency number is extracted from the card image, formatted, and

stored in MISC(1) by subroutine INTEG. If an end-of-data is encountered (ISW=1), the plot type is set to "no sweep" by MISC(2)=1 and subroutine PLOTIN returns to the calling program.

Otherwise the plot type is extracted from the card image by subroutine ALPHA and tested against a table of valid alphanumeric types. If no match is found, subroutine PLOTIN exits with an appropriate error condition. If a match is found, the plot type is encoded in MISC(2) and subroutine PLOTIN returns to the calling program.

TYPE=5 Order Card

The ORDER identifier is decoded by subroutine ALPHA and the order number is extracted from the card image, formatted, and stored in MISC(3) by subroutine INTEG. Subroutine PLOTIN returns to the calling program.

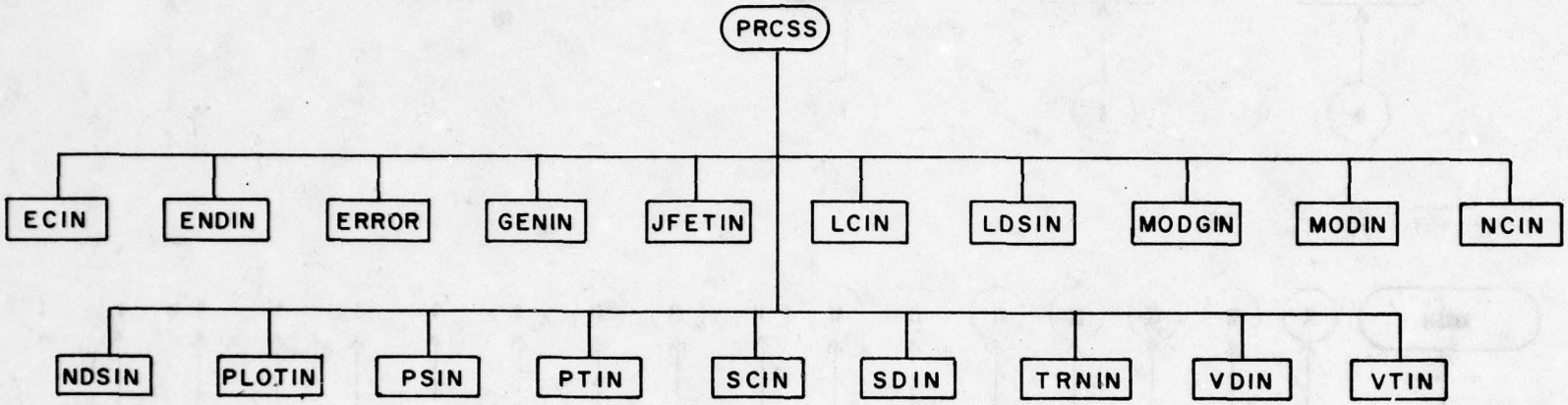
TYPE=6 Node Card

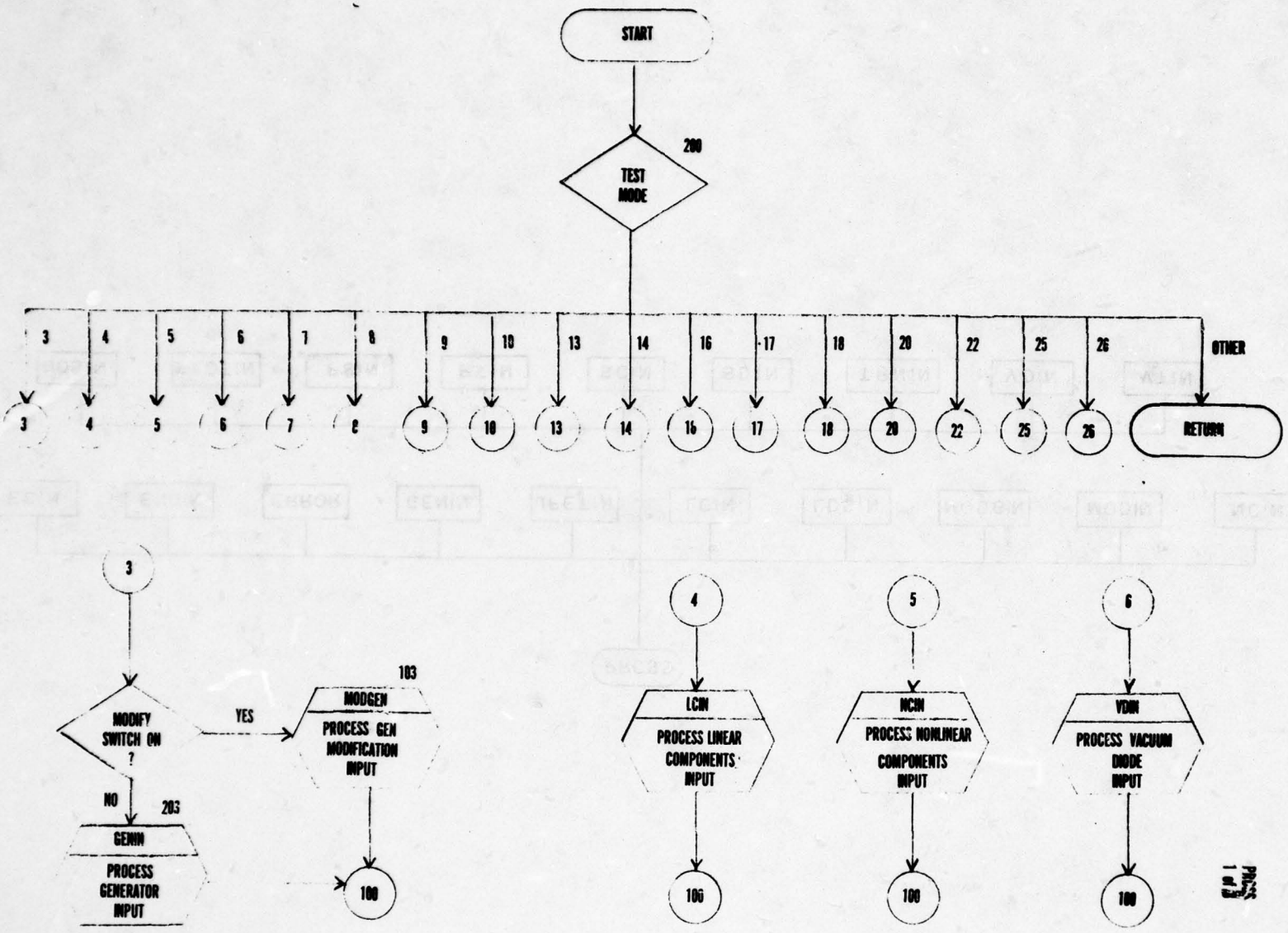
The NODE identifier is decoded by subroutine ALPHA and if an end-of-data is encountered (ISW=1), subroutine PLOTIN exits with an appropriate error condition. Otherwise the node number is extracted from the card image, formatted, and stored in MISC(4) by subroutine INTEG. Subroutine PLOTIN returns to the calling program.

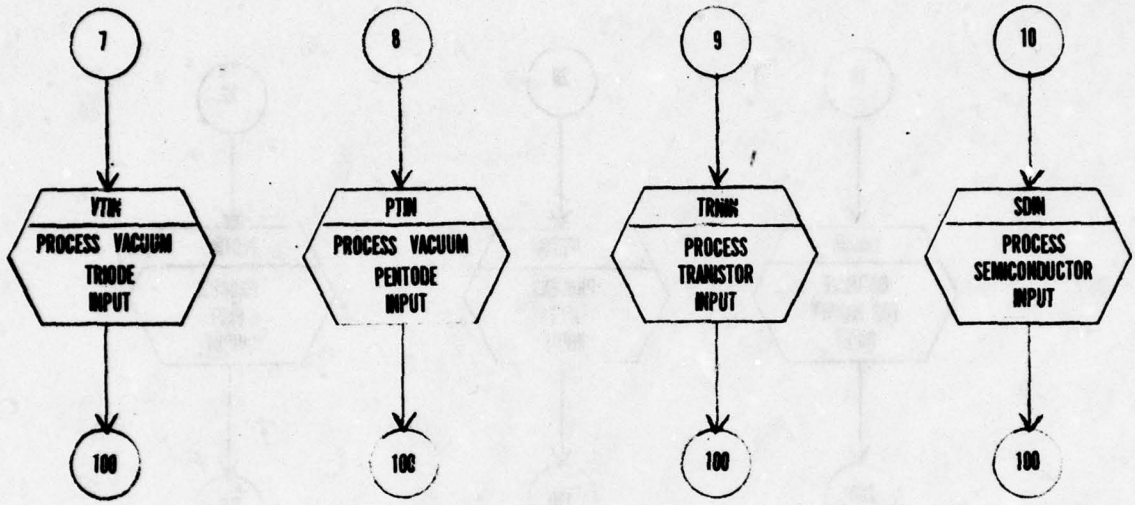
TYPE=15 Label Card

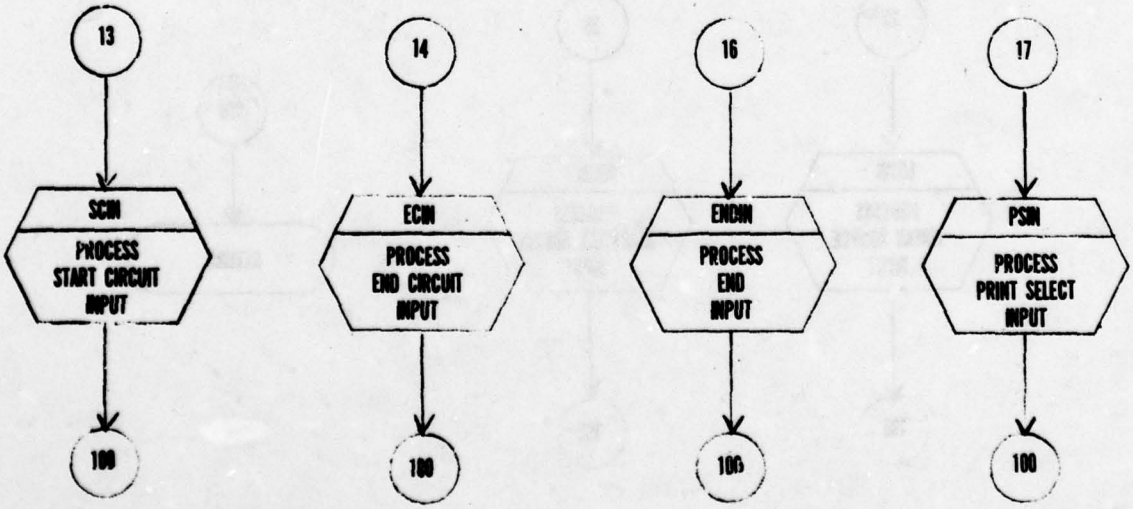
The 50-character label specification is extracted from the card image, stored in the common data buffer BUFF(1)-(50) and written to file 20 by subroutine DATAWR. The data file record number DATREC is updated and the data record length is incremented. After clearing the data buffer, subroutine PLOTIN returns to the calling program.

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


PRCSS
ECIN
ENDIN
ERROR
GENIN
JFETIN
LCIN
LDSIN
MODGIN
MODIN
NCIN
NDSIN
PLOTIN
PSIN
PTIN
SCIN
SDIN
TRNIN
VDIN
VTIN

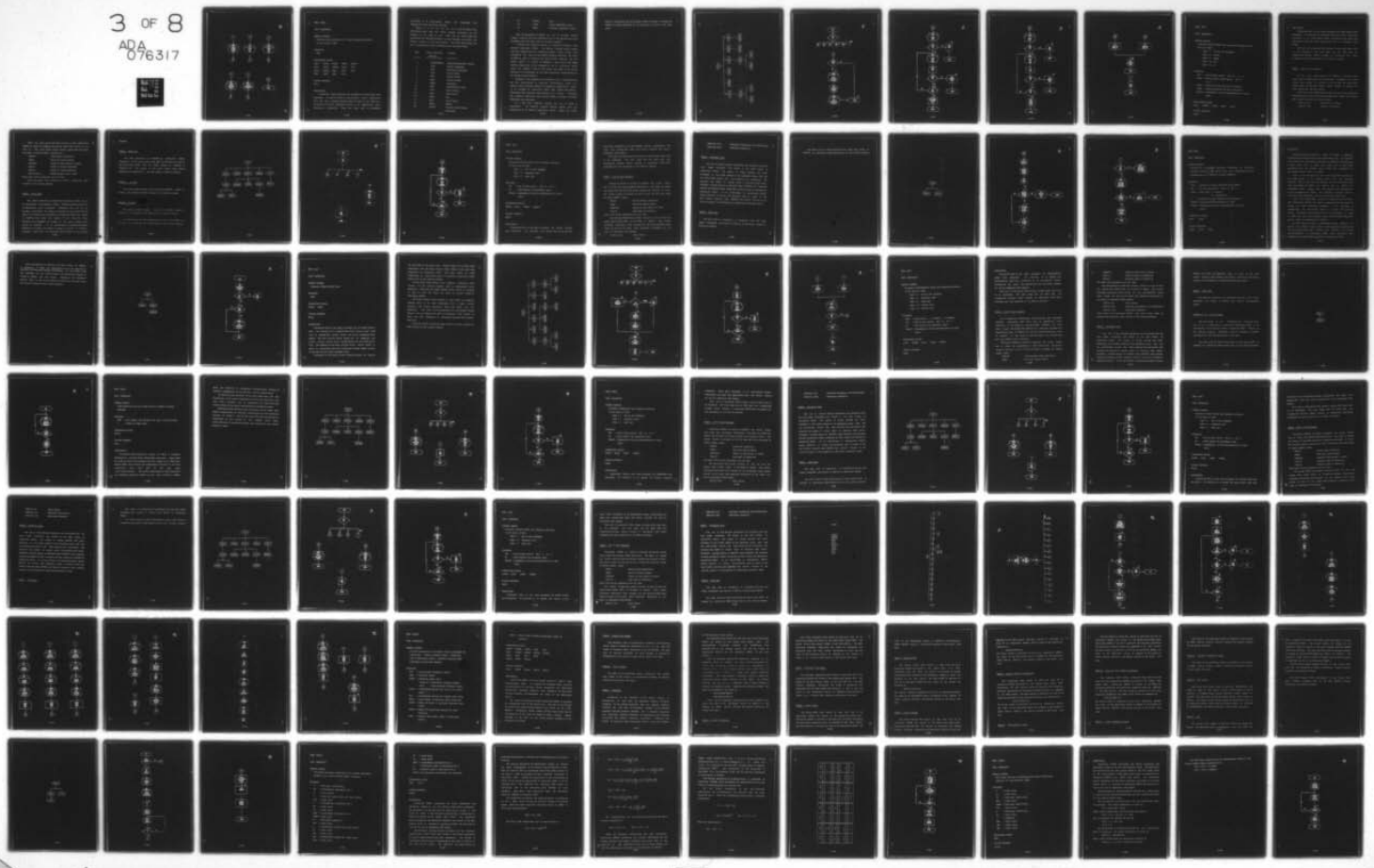

START
200
TEST MODE
3
4
5
6
7
8
9
10
13
14
16
17
18
20
22
25
26
OTHER
RETURN
3
MODIFY SWITCH ON ?
YES
NO
103
MODGEN
PROCESS GEN MODIFICATION INPUT
203
GENIN
PROCESS GENERATOR INPUT
100
4
LCIN
PROCESS LINEAR COMPONENTS INPUT
106
5
NCIN
PROCESS NONLINEAR COMPONENTS INPUT
100
6
VDIN
PROCESS VACUUM DIODE INPUT
100

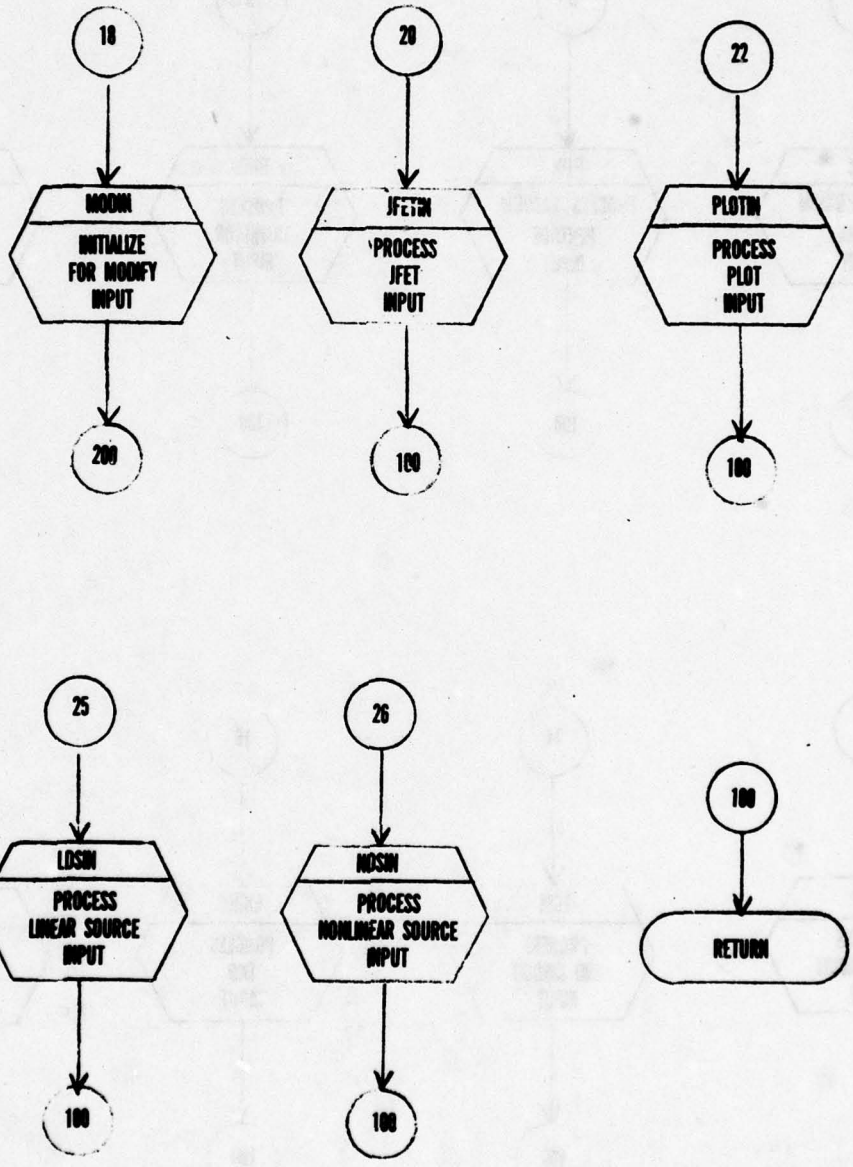

7
VTIN
PROCESS VACUUM TRIODE INPUT
100
8
PTIN
PROCESS VACUUM PENTODE INPUT
100
9
TRIN
PROCESS TRANISTOR INPUT
100
10
SDIN
PROCESS SEMICONDUCTOR INPUT
100


13
SCIN
PROCESS START CIRCUIT INPUT
100
14
ECIN
PROCESS END CIRCUIT INPUT
100
16
ENDIN
PROCESS END INPUT
100
17
PSIN
PROCESS PRINT SELECT INPUT
100

AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

3 OF 8
ADA 076317




18
MODIN
INITIALIZE FOR MODIFY INPUT
200
20
JFETIN
PROCESS JFET INPUT
100
22
PLOTIN
PROCESS PLOT INPUT
100
25
LOSIN
PROCESS LINEAR SOURCE INPUT
100
26
NOSIN
PROCESS NONLINEAR SOURCE INPUT
100
100
RETURN

NAME: PRCSS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Controls input processing of *-card sequences according to the value of MODE

VARIABLES:

NONE

SUBROUTINES CALLED:

ECIN	ENDIN	ERROR	GENIN	JFETIN
LCIN	LDSIN	MODGIN	MODIN	NCIN
NDSIN	PLOTIN	PSIN	PTIN	SCIN
SDIN	TRNIN	VDIN	VTIN	

CALLING PROGRAMS:

MAIN

DESCRIPTION:

Subroutine PRCSS controls the processing of NCAP input card sequences. It uses the MODE of the previous *-card transmitted from the main program through common storage as the index of a computed GO TO which transfers control to an appropriate input processing subroutine. There the input card is processed

according to an anticipated format and translated into appropriate data and driver records.

Modes 1, 2, 11, 12, 15, 19, 21, 23 and 24 imply error conditions since they are either created internally by the program or not used at all. When any of these modes are encountered by subroutine PRCSS, an error message is printed and control returns to the main program. The valid MODE values and their corresponding input processors are tabulated below:

MODE	Input Processing Routine	Comments
3	GENIN/MODGIN	Generator/Generator Modify
4	LCIN	Linear Components
5	NCIN	Nonlinear Components
6	VDIN	Vacuum Diode
7	VTIN	Vacuum Triode
8	PTIN	Vacuum Pentode
9	TRNIN	Transistor
10	SDIN	Semiconductor Diode
13	SCIN	Start Circuit
14	ECIN	End Circuit
16	ENDIN	End
17	PSIN	Print Select
18	MODIN	Modify
20	JFETIN	Junction Field Effect Transistor

22	PLOTIN	Plot
25	LDSIN	Linear Dependent Source
26	NDSIN	Nonlinear Dependent Source

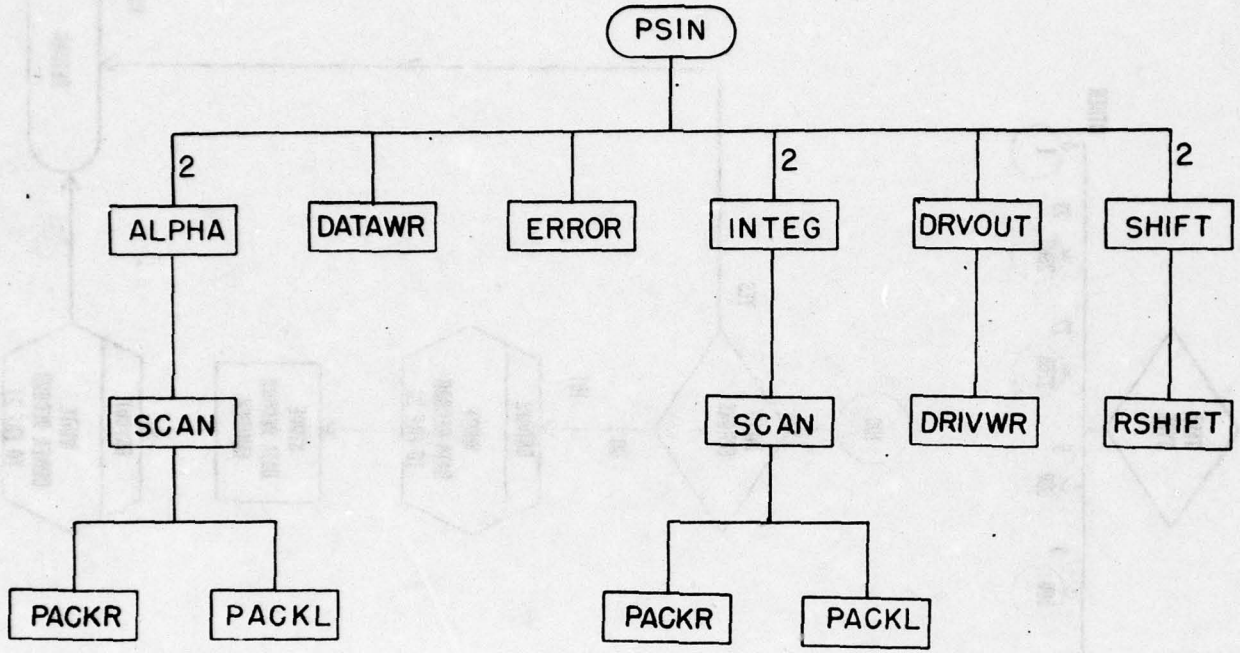
With the exception of MODE=3, 16, and 18 program control passes directly from the computed GO TO to the appropriate input processor and from there back to the main program.

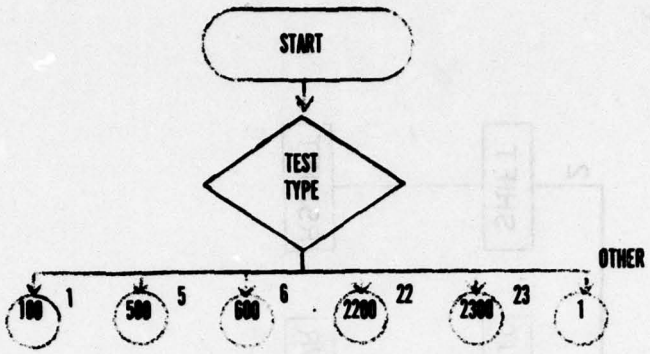
* MODIFY card sencences result in a slightly different path through subroutine PRCSS. For MODE=18, program control passes from the computed GO TO to subroutine MODIN. There, rather than processing the cards of the modify sequence directly, the value of MODE is reset to identify the device being modified and the modify switch is turned on (MODFY=1). Upon return from MODIN control passes back to the computed GO TO of subroutine PRCSS where the updated value of MODE causes the cards of the modify sequence to be processed by the input subroutine corresponding to the device being modified.

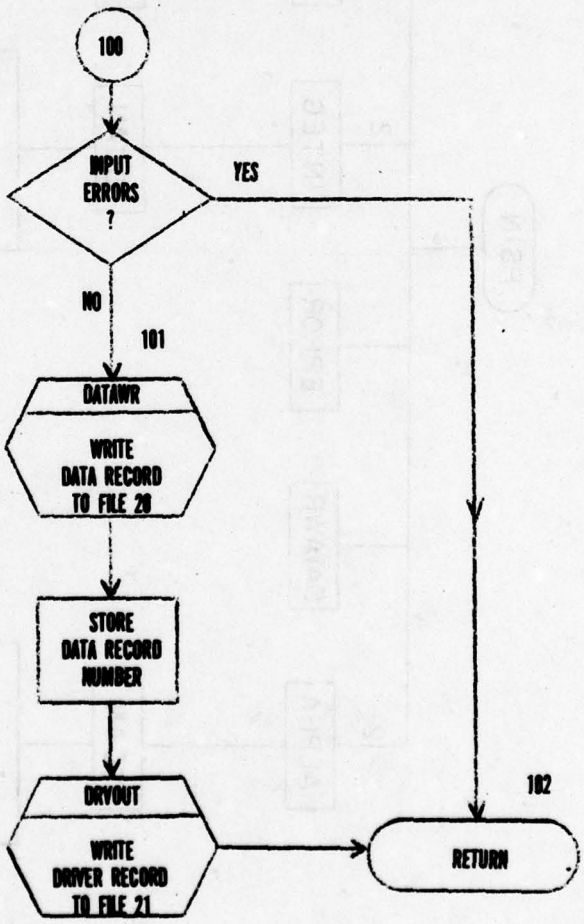
Because of the complexity of generator input specifications and the restrictions on generator modification, there is a special input processor MODGIN for generator modification cards. It is invoked by subroutine PRCSS when MODE=3 and MODFY=1, indicating that generator modification is in effect. Otherwise the standard generator input routine CENIN is called to process GENERATOR card sequences.

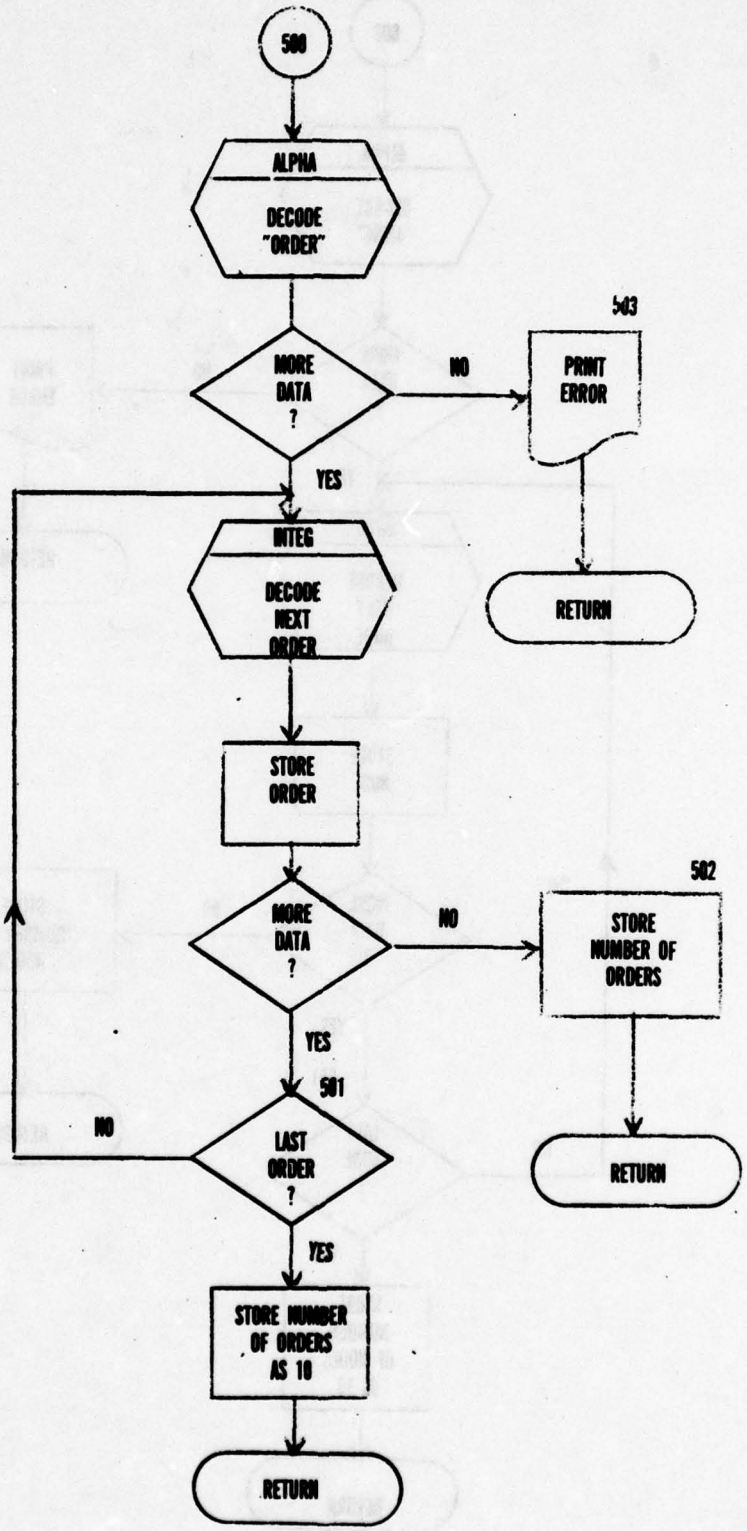
An * END card (MODE=16) signals the end of Phase 0 processing. For MODE=16, program control passes from the computed GO TO in PRCSS to subroutine ENDIN. There the input

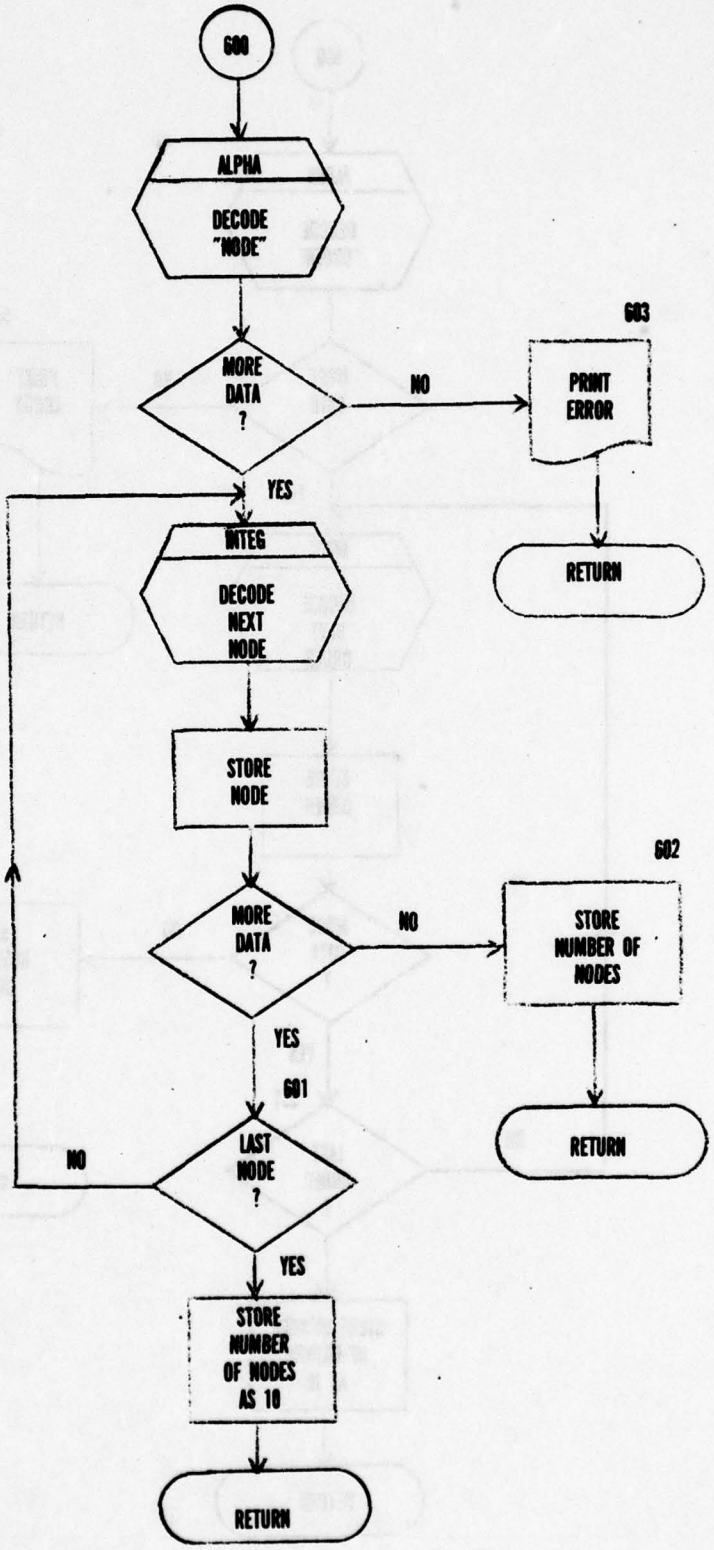
phase is terminated and the program either proceeds to subroutine PHASE1 or aborts depending on the presence of errors in the input deck.

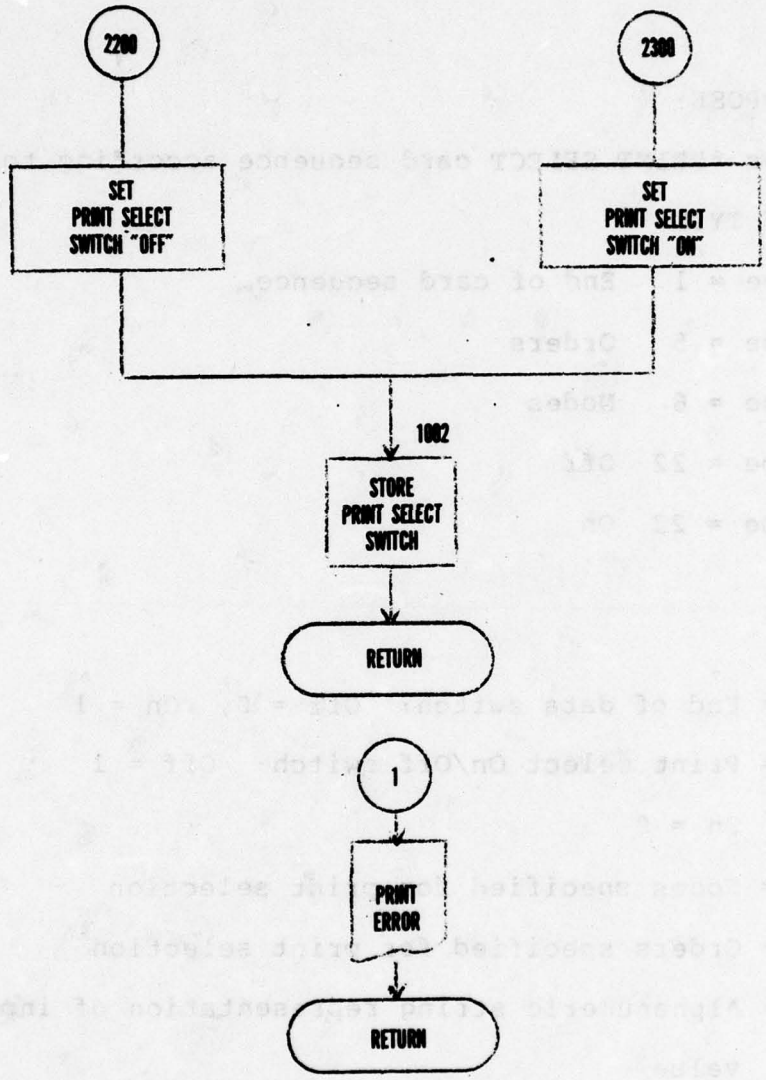

PSIN
2
2
2
ALPHA
DATAWR
ERROR
INTEG
DRVOUT
SHIFT
SCAN
SCAN
DRIVWR
RSHIFT
PACKR
PACKL
PACKR
PACKL


START
TEST TYPE
OTHER
100
1
500
5
600
6
2200
22
2300
23
1


100
INPUT ERRORS ?
YES
NO
101
DATAWR
WRITE DATA RECORD TO FILE 20
STORE DATA RECORD NUMBER
DRYOUT
WRITE DRIVER RECORD TO FILE 21
102
RETURN


500
ALPHA
DECODE
"ORDER"
503
MORE
DATA
?
NO
PRINT
ERROR
YES
INTEG
DECODE
NEXT
ORDER
RETURN
STORE
ORDER
502
MORE
DATA
?
NO
STORE
NUMBER OF
ORDERS
YES
501
NO
LAST
ORDER
?
RETURN
YES
STORE NUMBER
OF ORDERS
AS 10
RETURN


600
ALPHA
DECODE "MODE"
603
MORE DATA ?
NO
PRINT ERROR
YES
INTEG
DECODE NEXT NODE
RETURN
STORE NODE
602
MORE DATA ?
NO
STORE NUMBER OF NODES
YES
601
LAST NODE ?
NO
RETURN
YES
STORE NUMBER OF NODES AS 10
RETURN


2200
2300
SET
PRINT SELECT
SWITCH "OFF"
SET
PRINT SELECT
SWITCH "ON"
1002
STORE
PRINT SELECT
SWITCH
RETURN
1
PRINT
ERROR
RETURN

NAME: PSIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *PRINT SELECT card sequence according to the value of TYPE:

Type = 1 End of card sequence

Type = 5 Orders

Type = 6 Nodes

Type = 22 Off

Type = 23 On

VARIABLES:

ISW = End of data switch: Off = 0, On = 1

NLIST = Print select On/Off switch: Off = 1
On = 0

NODES = Nodes specified for print selection

ORDRS = Orders specified for print selection

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

ALPHA DATAWR DRVOUT ERROR INTEG

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine PSIN is the input processor for PRINT SELECT card sequences. Its function is to decode the print select input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine PSIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine PSIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

If the input error switch is on (JAPT=1), subroutine PSIN exits with an appropriate error condition. Otherwise the print select data record is written to file 20 from the common data buffer BUFF, and the data record number DATREC is updated to point beyond the new data record.

Since the print select feature does not cause any internally generated data to be transmitted between NCAP phases, the 20-word data record contains only input values:

BUFF(1)-(10)	Orders to be Printed
BUFF(11)-(20)	Nodes to be Printed

After the data record has been written to disk, subroutine DRVOUT is called to complete the driver record and write it to file 21. The print select driver record, taken from the first ten words of global common, consists of:

MODE=17	Print Select Identifier
STADD	Data File Record Number
LNGTH=20	Length of Data Record in Words
MISC(1)	Number of Orders Specified
MISC(2)	Number of Nodes Specified
MISC(3)=0 or 1	ON/OFF switch: 0=on, 1=off

Other MISC driver parameters are not used.

After returning from subroutine DRVOUT, subroutine PSIN returns to the calling program.

TYPE=5 Order Card

The ORDER identifier is decoded by subroutine ALPHA, and if an end-of-data is encountered (ISW=1), subroutine PSIN exits with an appropriate error condition. Otherwise from one to ten integers specifying the orders of analysis to be printed during Phase 4 are decoded and formatted by subroutine INTEG and stored in ORDRS(1)-(10) under the control of the index KZ. After decoding, each ORDRS(KZ) is placed in the print select data record at BUFF(KZ). If an end-of-data is encountered before ORDRS(10) is filled, the number of orders is stored in MISC(1). Otherwise MISC(1)=10 and subroutine PSIN returns to the calling

program.

TYPE=6 Node Card

The NODE identifier is decoded by subroutine ALPHA. Processing of the print select node data is identical to that of the order data except that the input values are decoded to NODES(1)-(10) and stored in the print select data record beginning at BUFF(KZ+10). The node count is saved in MISC(2).

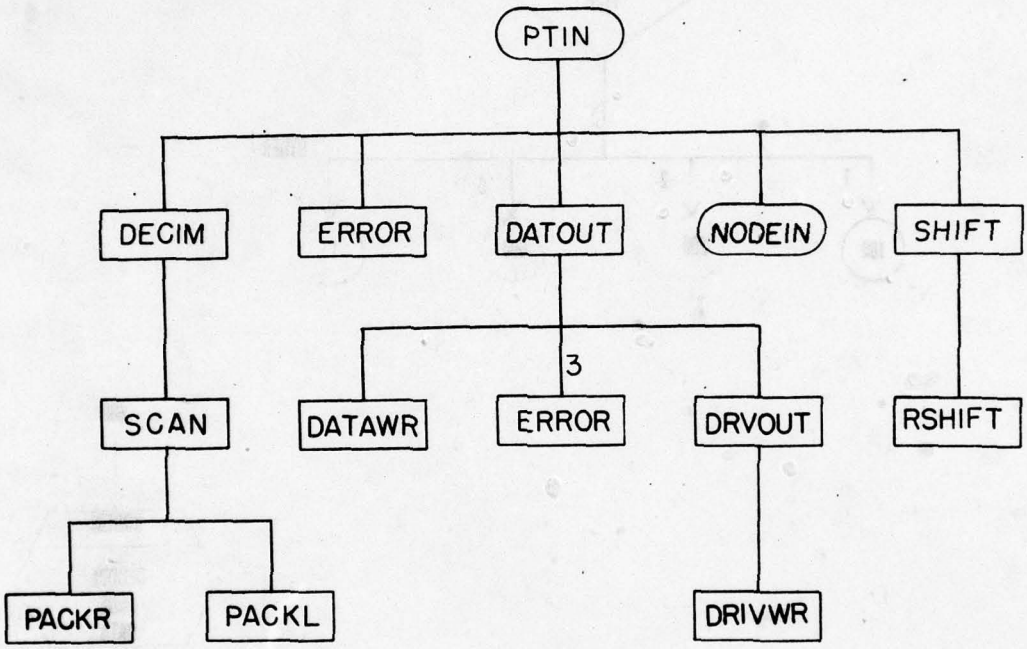
TYPE=22 Off Card

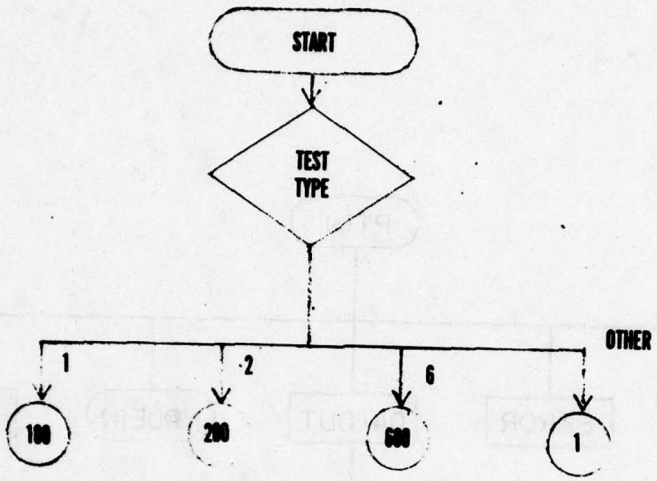
The print select switch is set to OFF by NLIST=1, saved in MISC(3), and subroutine PSIN returns to the calling program.

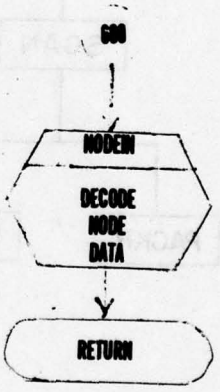
TYPE=23 On Card

The print select switch is set to ON by NLIST=0, saved in MISC(3), and subroutine PSIN returns to the calling program.

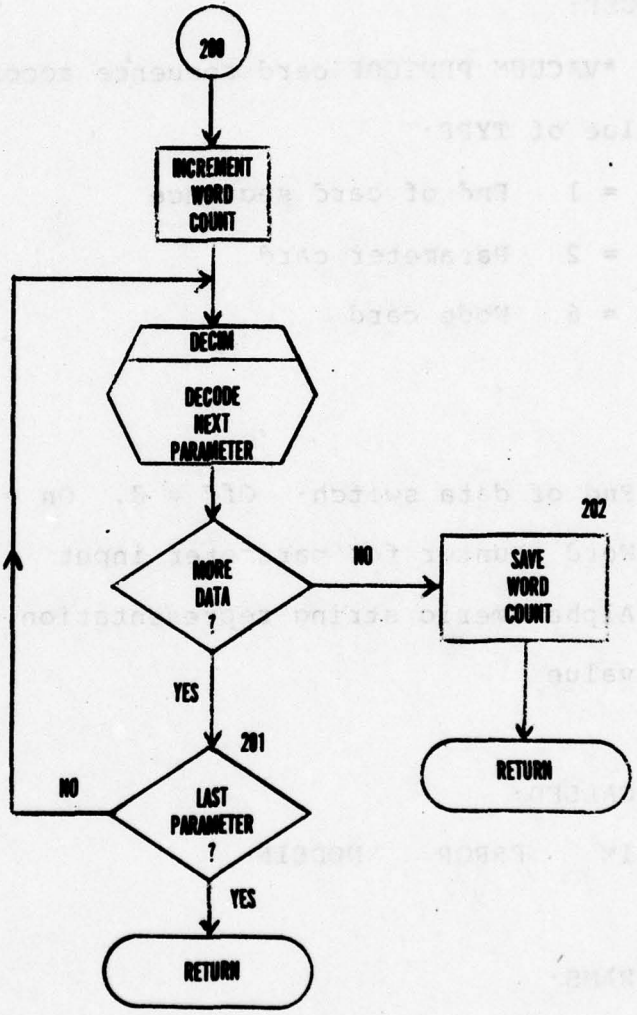
Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


PTIN
DECIM
ERROR
DATOUT
NODEIN
SHIFT
3
SCAN
DATAWR
ERROR
DRVOUT
RSHIFT
PACKR
PACKL
DRIVWR


START
TEST
TYPE
1
2
6
OTHER
100
200
600
1


600
MODEM
DECODE
MODE
DATA
RETURN


1
PRINT
ERROR
RETURN


200
INCREMENT WORD COUNT
DECOM
DECODE NEXT PARAMETER
MORE DATA ?
NO
202
SAVE WORD COUNT
RETURN
YES
201
NO
LAST PARAMETER ?
YES
RETURN

NAME: PTIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *VACUUM PENTODE card sequence according to the value of TYPE:

- Type = 1 End of card sequence
- TyPe = 2 Parameter card
- Type = 6 Node card

VARIABLES:

ISW = End of data switch: Off = 0, On = 1

NW = Word counter for parameter input

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

DATOUT DECIM ERROR NODEIN

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine PTIN is the input processor for VACUUM PENTODE card sequences. Its function is to decode the vacuum pentode

input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine PTIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine PTIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

Subroutine DATCUT is called to complete the driver record and to close the vacuum pentode definition. The input is tested for errors and the data and driver records are written to disk. The driver record is written to file 21 from the first ten words of global common, where:

MODE=8	Vacuum Pentode Identifier
STADD	Data File Record Number
LNGTH=107	Length of Data Record in Words
MISC(3)	Base Node of Connection

Other MISC driver parameters are not used.

The vacuum pentode data record, written to file 20 from the common data buffer BUFF, is 107 words in length. This record allocates sufficient disk storage for the vacuum pentode input values as well as all other data generated internally by its model in subsequent NCAP phases.

BUFF(1)-(12)	Input Values

BUFF(13)-(75)	Nonlinear Parameters and Coefficients
BUFF(76)-(107)	Admittance Submatrix

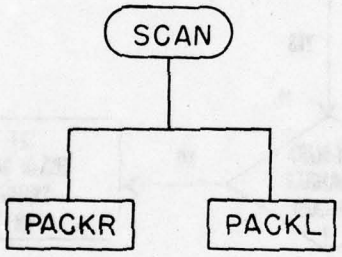
TYPE=2 Parameter Card

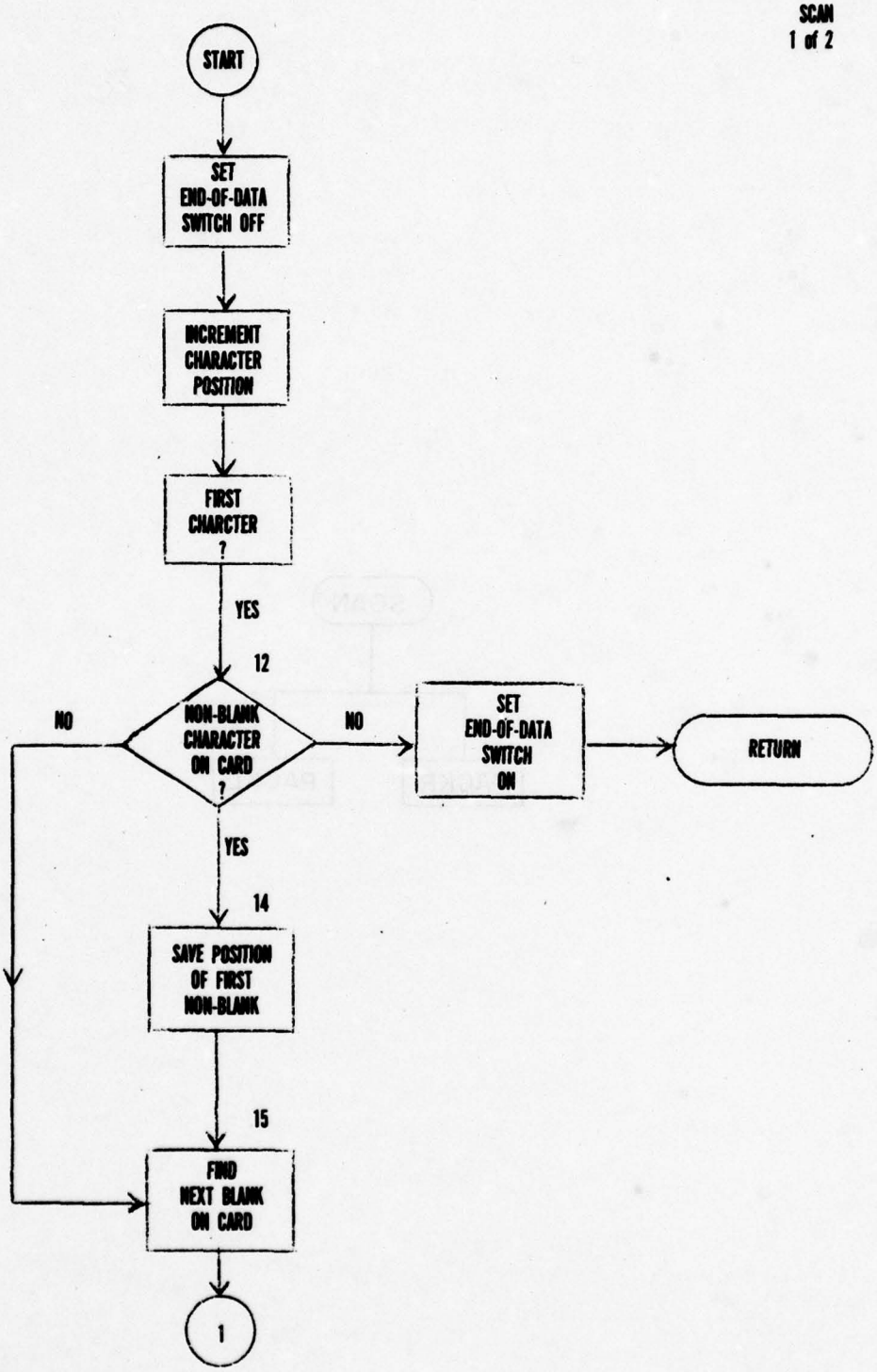
From one to twelve decimal parameters are extracted from the card image, formatted, and stored in the data buffer by subroutine DECIM. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of values input on previous data cards. Parameter storage begins at BUFF(NW) where NW=NXT+1 and proceeds through successive buffer locations as data values are decoded by subroutine DECIM. If an end-of-data is encountered (ISW=1) before BUFF(12) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation of additional parameter cards.

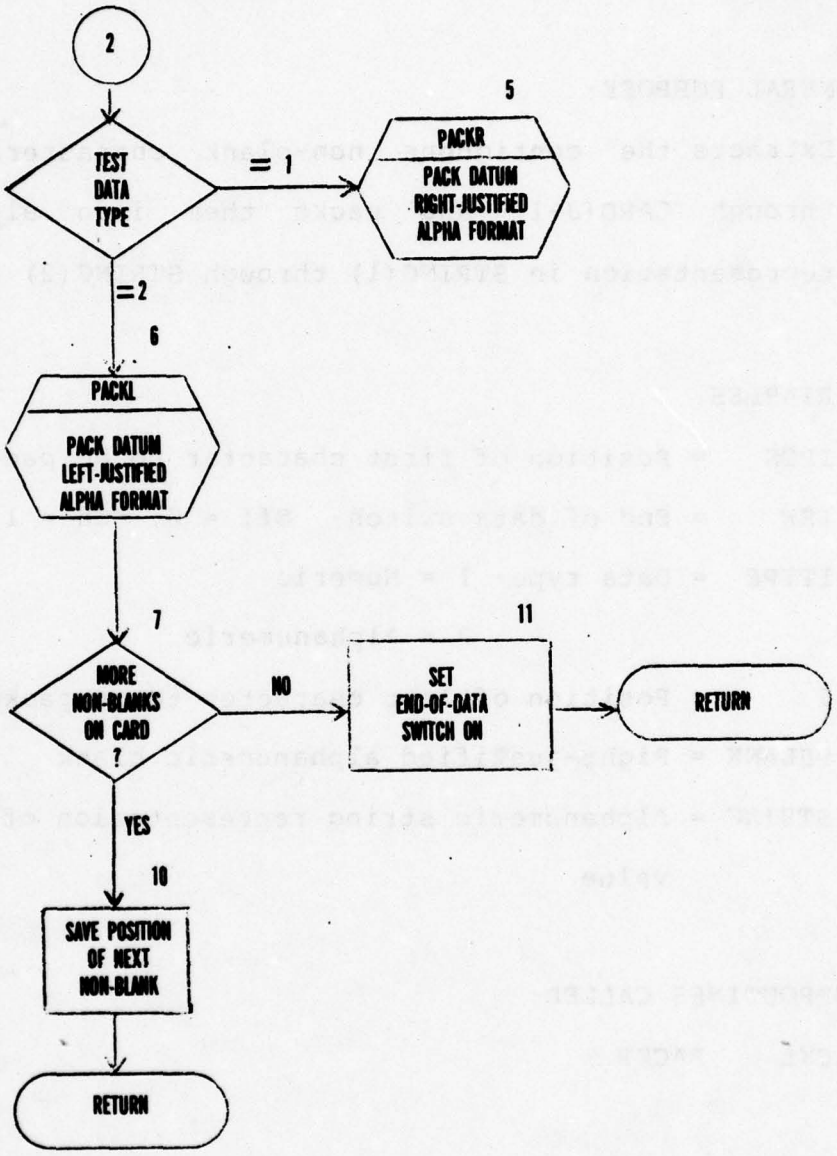
TYPE=6 Node Card

The base node of connection is extracted from the card image, formatted, and stored in MISC(3) of the driver record by subroutine NODEIN.

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


SCAN
PACKR
PACKL


START
SET END-OF-DATA SWITCH OFF
INCREMENT CHARACTER POSITION
FIRST CHARCTER ?
YES
12
NON-BLANK CHARACTER ON CARD ?
NO
NO
SET END-OF-DATA SWITCH ON
RETURN
YES
14
SAVE POSITION OF FIRST NON-BLANK
15
FIND NEXT BLANK ON CARD
1


2
TEST DATA TYPE
= 1
5
PACKR
PACK DATUM RIGHT-JUSTIFIED ALPHA FORMAT
=2
6
PACKL
PACK DATUM LEFT-JUSTIFIED ALPHA FORMAT
7
MORE NON-BLANKS ON CARD ?
NO
11
SET END-OF-DATA SWITCH ON
RETURN
YES
10
SAVE POSITION OF NEXT NON-BLANK
RETURN

NAME: SCAN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Extracts the contiguous non-blank characters in CARD(IPOS) through CARD(J-1) and packs them into alphanumeric string representation in STRING(1) through STRING(2)

VARIABLES:

IPOS = Position of first character to be packed

ISW = End of data switch: Off = 0, On = 1

ITYPE = Data type: 1 = Numeric
2 = Alphanumeric

J = Position of last character to be packed + 1

RBLANK = Right-justified alphanumeric blank

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

PACKL PACKR

CALLING PROGRAMS:

ALPHA DECIM INTEG

DESCRIPTION:

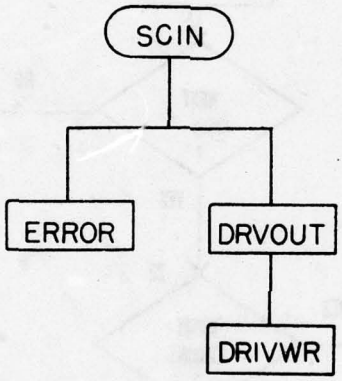
Subroutine SCAN belongs to a group of Phase Ø character manipulation routines which format NCAP input data. Its function is to extract the contiguous alphanumeric characters constituting a data value from an input card image and to pack those characters into the integer array STRING for formatting by subroutines ALPHA, DECIM, and INTEC.

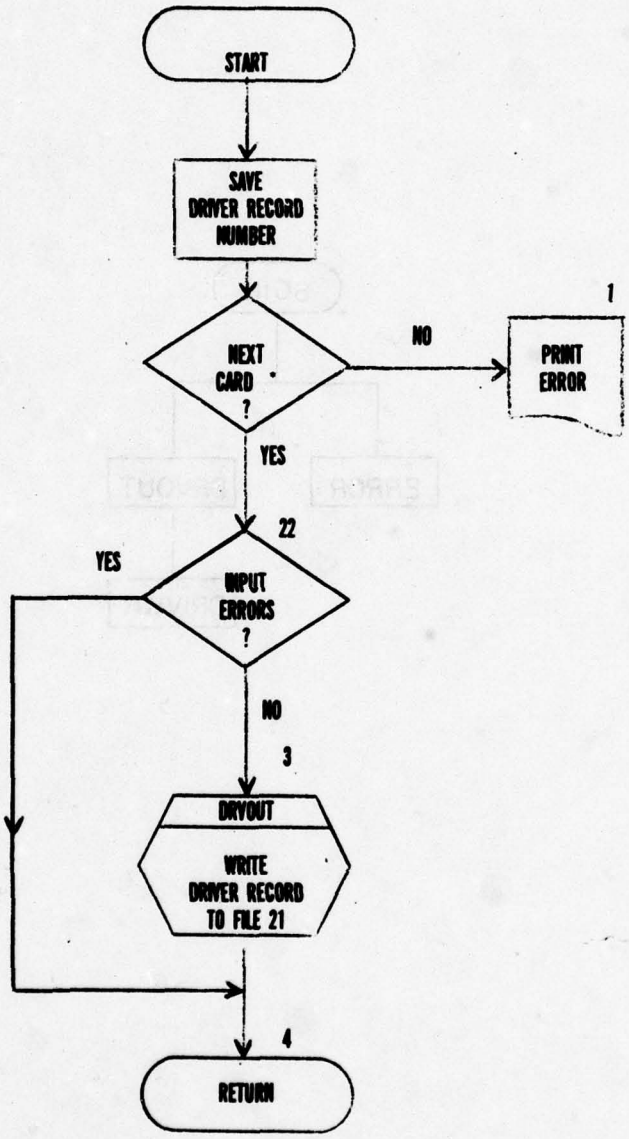
Input to the subroutine is the right-justified alphanumeric card image stored in the integer array card, the character position IPOS at which the scan begins, and the anticipated data type represented by ITYPE=1 for numeric data or ITYPE=2 for alphabetic data. Output from the routine is the alphanumeric string representation STRING, the character position IPOS at which the scan terminated, and the end-of-data switch ISW.

First the end-of-data switch is initialized off by ISW=Ø. Then starting at CARD(I) the image is scanned column-by-column to locate the first non-blank character. If no non-blank is encountered, the end-of-data switch is turned "on" and exit occurs. Otherwise the position of the first non-blank is saved in IPOS and the scan proceeds across the non-blank characters until a blank is encountered at CARD(J).

At this point the datum in CARD(IPOS) through CARD(J-1) is packed into STRING either right or left-justified according to the anticipated data type. Numeric data (ITYPE=1) are packed right-justified by subroutine PACKR, while alphabetic data (ITYPE=2) are packed left-justified by subroutine PACKL.

After the packing is complete, the scan resumes at CARD(J) to determine if there are additional data on the card and to position the character pointer accordingly. In the event that the remainder of the card is blank, the end-of-data switch is turned on (ISW=Ø) and exit occurs. Otherwise the character pointer IPOS is set to the starting location of the next datum and control returns to the calling program.


SCIN
ERROR
DRVOUT
DRIVWR


START
SAVE DRIVER RECORD NUMBER
NEXT CARD ?
NO
1
PRINT ERROR
YES
22
YES
INPUT ERRORS ?
NO
3
DRYOUT
WRITE DRIVER RECORD TO FILE 21
4
RETURN

NAME: SCIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *START CIRCUIT card

VARIABLES:

NONE

SUBROUTINES CALLED:

DRVOUT ERROR

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine SCIN is the input processor for the START CIRCUIT card. Its function is to translate the start circuit input card into an appropriate driver record for use by subsequent NCAP phases. The start circuit driver record and its companion end circuit driver record serve as delimiters for the NCAP driver file. The address of the start circuit driver record SCREC is saved by subroutine SCIN and transmitted through common storage to the end circuit input processor ECIN.

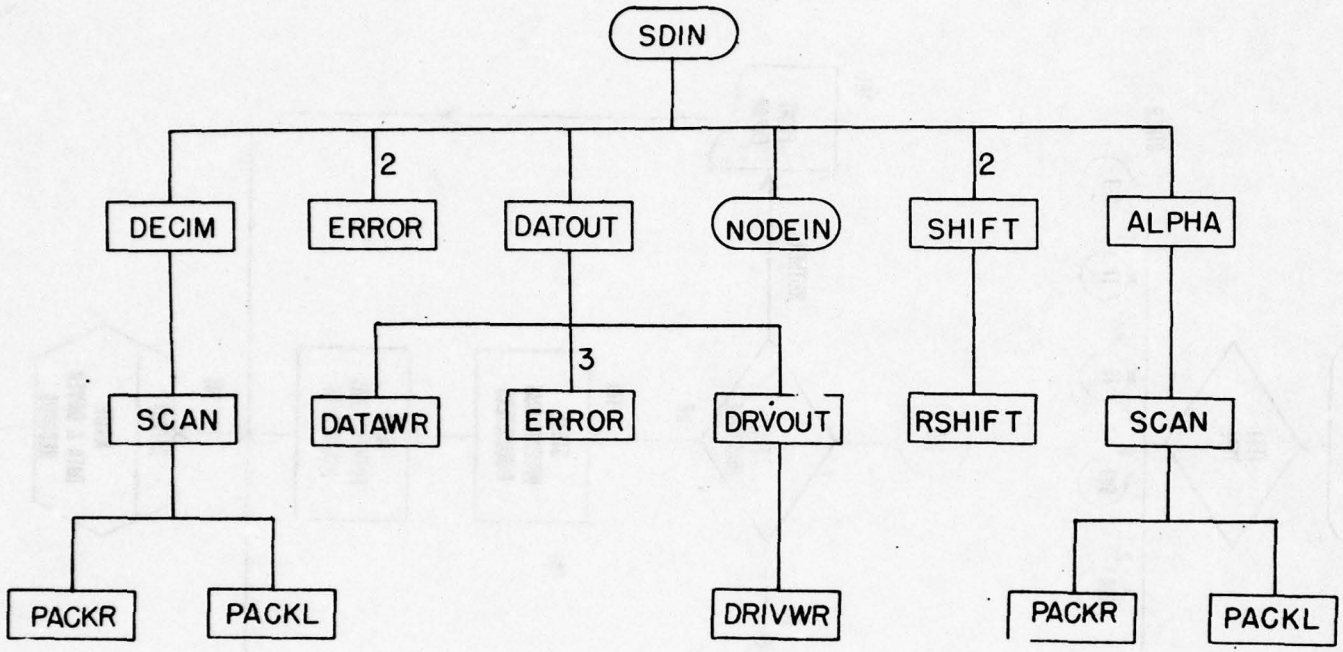
Processing of the start circuit function begins by testing

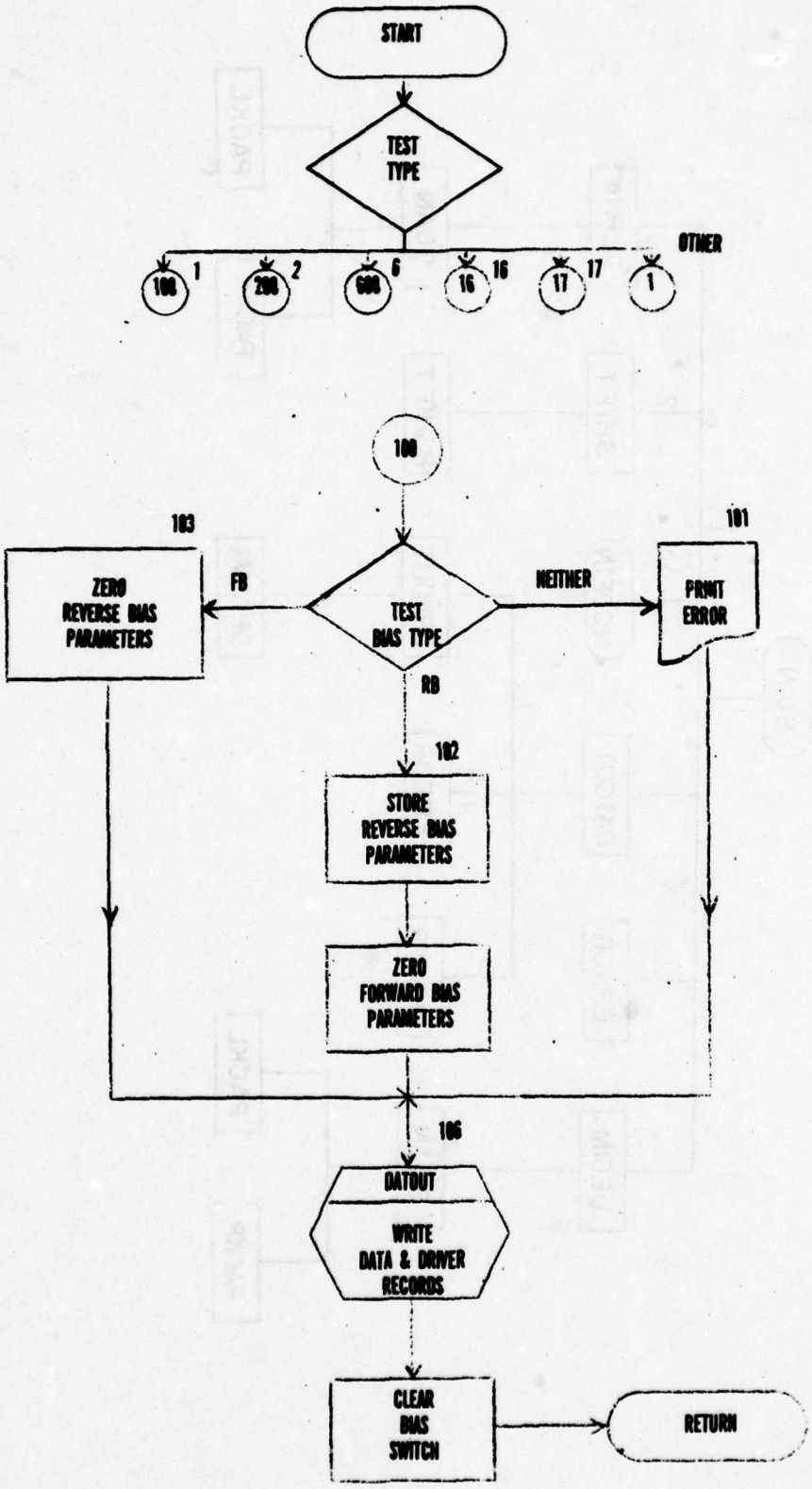
the TYPE code of the input card. Because there are no data cards associated with the start circuit input, TYPE=1 is the only code recognized by subroutine SCIN. Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.

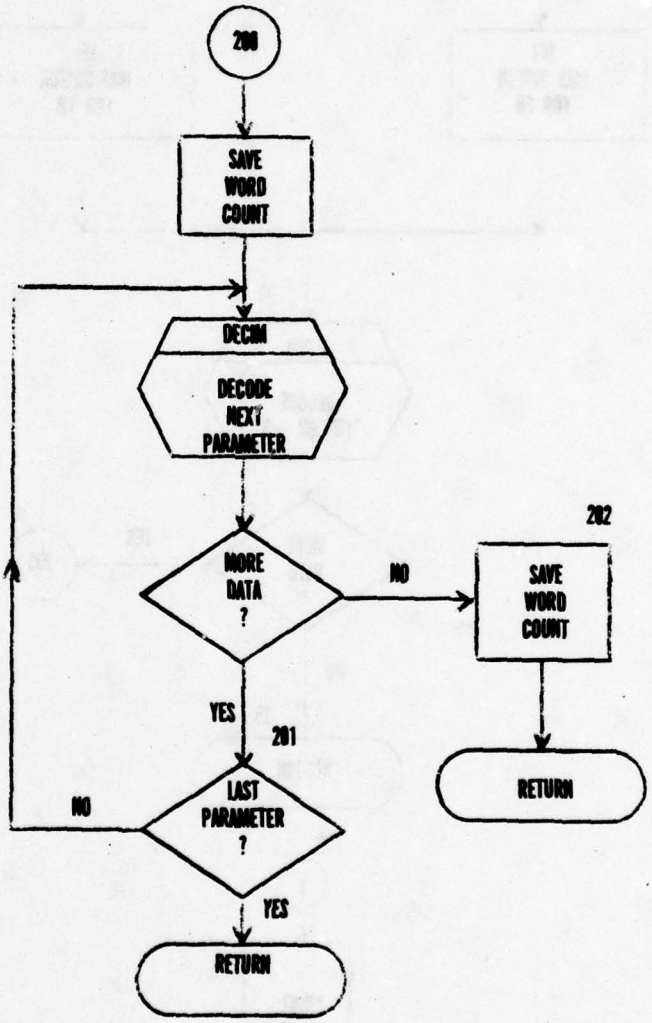
If the input error switch is on (JABT=1), subroutine SCIN returns to the calling program with an appropriate error condition. Otherwise the START CIRCUIT driver record is written to file 21 from the first ten words of global common by subroutine DRVOUT.

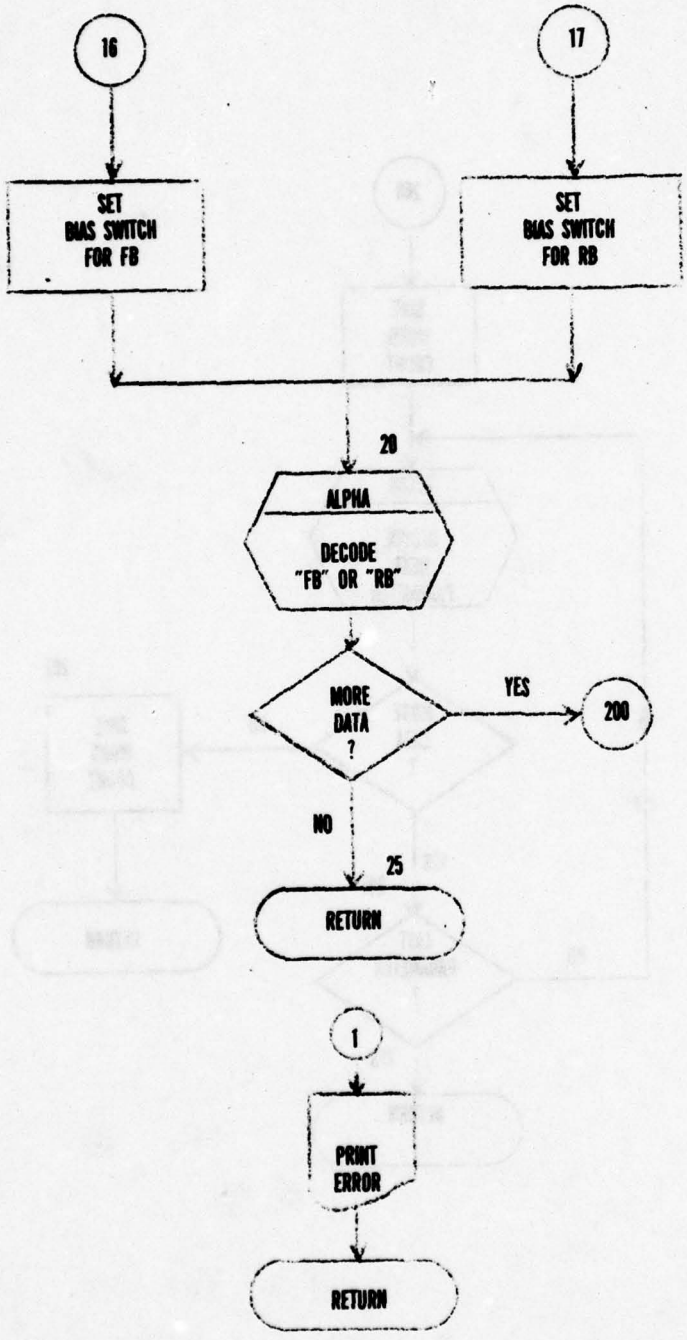
The start circuit driver record is identified by MODE=13. Because there is no data associated with a start circuit function, the driver parameters STADD and LNGTH are not applicable. The MISC driver parameters are not defined during Phase Ø, but are defined and used in subsequent NCAP phases to store the disk addresses of admittance matrices and transfer functions.

After the driver record has been written to disk, subroutine SCIN returns to the calling program.


SDIN
2
2
DECIM
ERROR
DATOUT
NODEIN
SHIFT
ALPHA
3
SCAN
DATAWR
ERROR
DRVOUT
RSHIFT
SCAN
PACKR
PACKL
DRIVWR
PACKR
PACKL


START
TEST TYPE
OTHER
100 1
200 2
600 6
16 16
17 17
1
100
TEST BIAS TYPE
FB
NEITHER
RB
103
ZERO REVERSE BIAS PARAMETERS
101
PRINT ERROR
102
STORE REVERSE BIAS PARAMETERS
ZERO FORWARD BIAS PARAMETERS
106
DATOUT
WRITE DATA & DRIVER RECORDS
CLEAR BIAS SWITCH
RETURN


200
SAVE WORD COUNT
DECIM
DECODE NEXT PARAMETER
MORE DATA ?
NO
202
SAVE WORD COUNT
RETURN
YES
201
LAST PARAMETER ?
NO
YES
RETURN


16
17
SET
BIAS SWITCH
FOR FB
SET
BIAS SWITCH
FOR RB
20
ALPHA
DECODE
"FB" OR "RB"
MORE
DATA
?
YES
200
NO
25
RETURN
1
PRINT
ERROR
RETURN

NAME: SDIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *SEMICONDUCTOR DIODE card sequence according to the value of TYPE:

Type = 1	End of card sequence
Type = 2	Parameter card
Type = 6	Node card
Type = 16	Forward Bias
Type = 17	Reverse Bias

VARIABLES:

IBSW	= Bias switch; 1 = Forward, 2 = Reverse
ISW	= End of data switch. Off = 0, On = 1
NW	= Word counter for parameter input
STRING	= Alphanumeric string representation of input value

SUBROUTINES CALLED:

ALPHA DATOUT DECIM ERROR NODEIN

CALLING PROGRAMS:

PRCSS

DESCRIPTION·

Subroutine SDIN is the input processor for SEMICONDUCTOR DIODE card sequences. Its function is to decode the semiconductor input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine SDIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine SDIN which processes the card according to its TYPE as follows·

TYPE=1 End of Card Sequence

If no forward or reverse-bias specification was processed (IBSW=Ø), subroutine SDIN exits with an appropriate error condition. If the diode is reverse-biased (IBSW=2), the four input values are moved from BUFF(1)-(4), which are reserved for forward-bias data, to BUFF(5)-(8) and the forward-bias data area is cleared. If the diode is forward-biased, the reverse-bias data area BUFF(5)-(8) is cleared.

Subroutine DATOUT is called to complete the driver record and to close the semiconductor diode definition. The driver record is written to file 21 from the first ten words of global common, where:

MODE=1Ø	Semiconductor Diode Identifier
STADD	Data File Record Number

LNGTH=32	Length of Data File in Words
MISC(3)	Positive Node of Connection
MISC(4)	Negative Node of Connection

The other MISC parameters are not used.

The semiconductor diode data record, written to file 20 from the common data buffer BUFF, is 32 words in length. This record allocates sufficient disk storage for the semiconductor diode input values as well as all other data generated internally by its model in subsequent NCAP phases:

BUFF(1)-(8)	Input Values
BUFF(9)-(30)	Nonlinear Parameters and Coefficients
BUFF(31)-(32)	Admittance Submatrix

After return from subroutine DATOUT, the bias switch IBSW is cleared and subroutine SDIN returns to the calling program.

TYPF=2 Parameter Card

From one to four decimal parameters are extracted from the card image, formatted, and stored in the data buffer by subroutine DECIM. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of values input on previous data cards. Parameter storage begins at BUFF(NW) where NW=NXT+1 and proceeds through successive buffer locations as data values are decoded by subroutine DECIM. If an end-of-data is encountered(ISW=1) before

BUFF(4) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation of additional parameter cards.

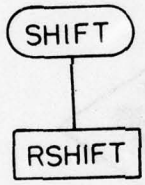
TYPE=6 Node Card

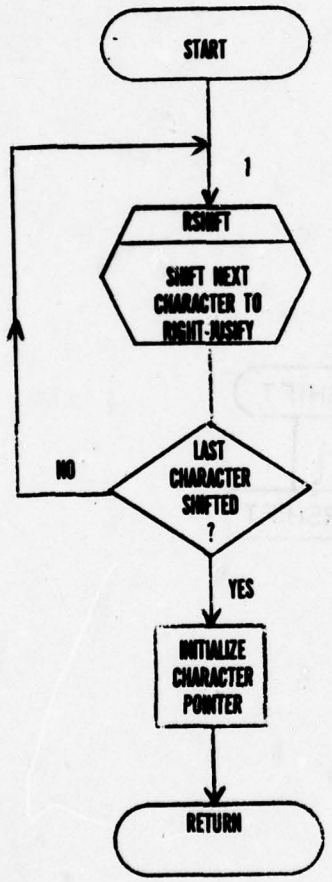
The nodes of connection are extracted from the card image, formatted, and stored in MISC(3) and MISC(4) by subroutine NODEIN.

TYPE=16 or 17 FB or RB Card

The bias switch is set: 1=forward-bias, 2=reverse-bias. The FB or RB identifier is decoded by subroutine ALPHA. If an end-of-data is encountered (ISW=1), subroutine SDIN returns to the calling program. Otherwise control transfers to TYPE=2 processing for the remaining data on the card.

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


SHIFT
RSHIFT


START
1
RSHIFT
SHIFT NEXT CHARACTER TO RIGHT-JUSIFY
LAST CHARACTER SHIFTED ?
NO
YES
INITIALIZE CHARACTER POINTER
RETURN

NAME: SHIFT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Right-justifies the card image stored in CARD(1) through CARD(80)

VARIABLES:

CARD = Card image, one character per word, left-justified

I = Index for CARD array

SUBROUTINES CALLED:

RSHIFT

CALLING PROGRAMS:

MAIN

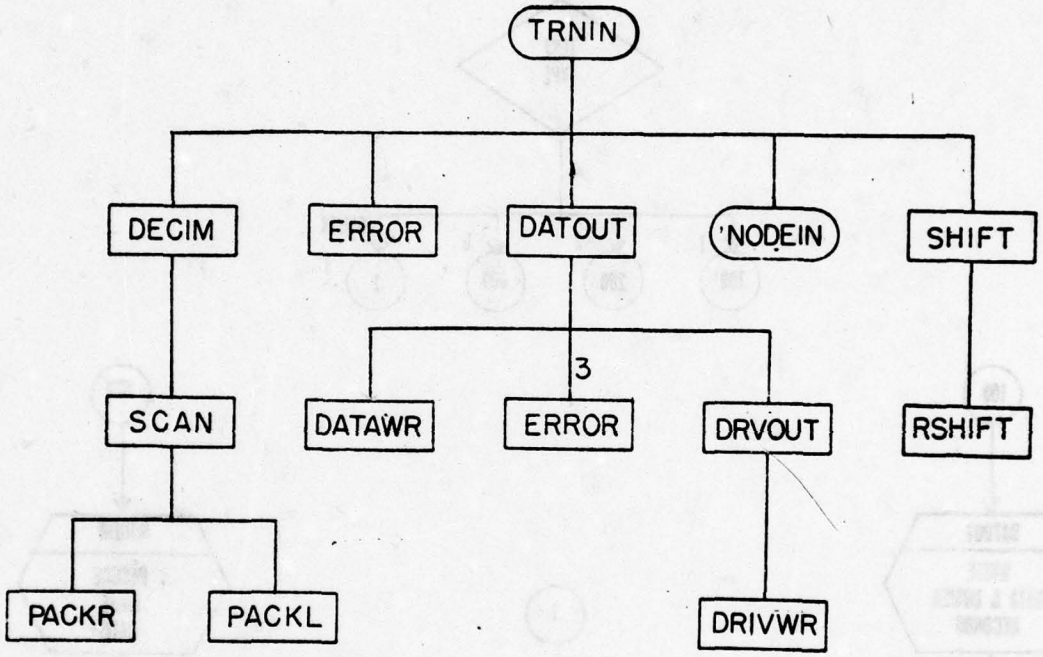
DESCRIPTION:

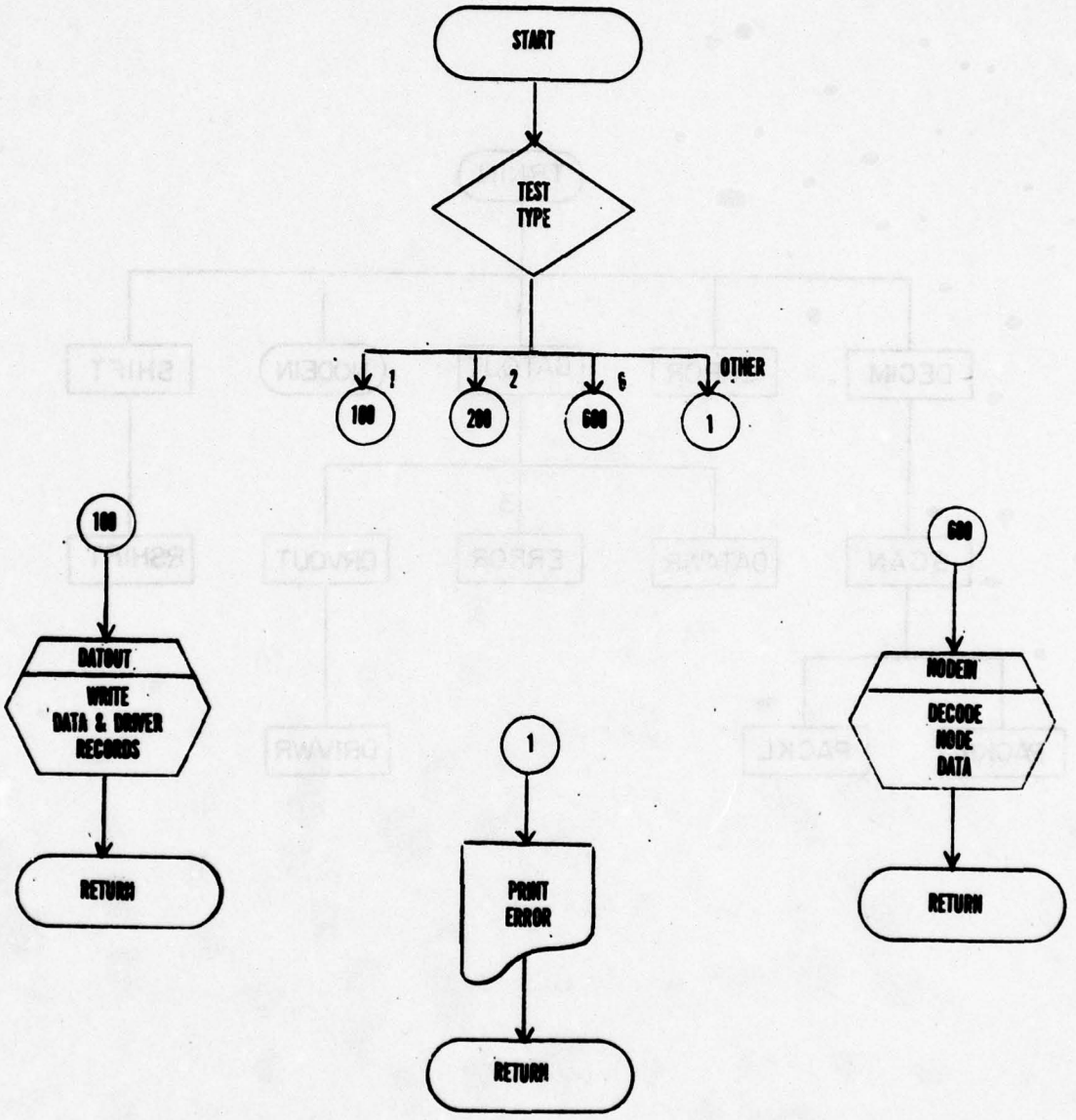
Subroutine SHIFT belongs to a group of Phase 0 character manipulation routines which format NCAP input data. Input cards are read by NCAP's main program into the integer array CARD under FORMAT (80A1) which places one alphanumeric character in the most significant bits each word of the card image (left-justification). The binary equivalents of such characters are frequently negative numbers or very large positive numbers

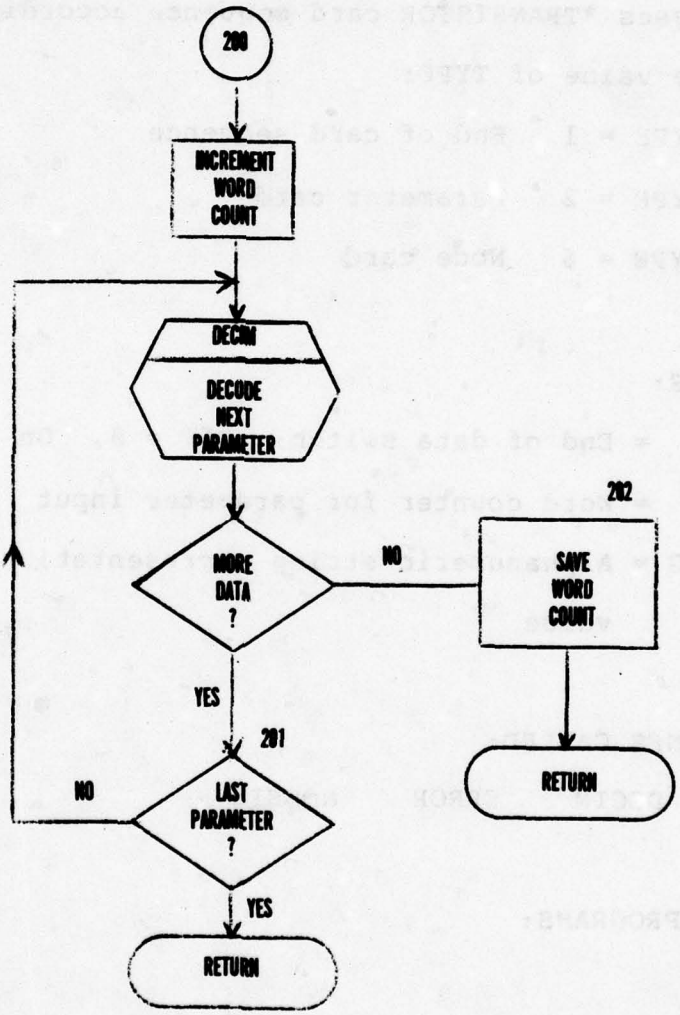
which are difficult to manipulate arithmetically because of overflow, propagation of the sign bit, and end-around carry.

By shifting each character of the card image from the most significant to the least significant bits and zero-filling to the left, every character can be represented as a small positive integer which can be easily manipulated by arithmetic means.

Subroutine SHIFT performs this conversion on the input card image in preparation for character manipulation by the formatting routines of Phase 0. The 80 left-justified card characters are transmitted to the routine in the integer array CARD, right-justified by subroutine RSHIFT, and returned to the calling program in CARD.


TRNIN
DECIM
ERROR
DATOUT
'NODEIN
SHIFT
3
SCAN
DATAWR
ERROR
DRVOUT
RSHIFT
PACKR
PACKL
DRIVWR


START
TEST TYPE
1
2
6
OTHER
100
200
600
1
100
DATOUT
WRITE DATA & DRIVER RECORDS
RETURN
1
PRINT ERROR
RETURN
600
MODEIN
DECODE MODE DATA
RETURN


200
INCREMENT WORD COUNT
DECRM
DECODE NEXT PARAMETER
MORE DATA ?
NO
202
SAVE WORD COUNT
YES
201
LAST PARAMETER ?
NO
RETURN
YES
RETURN

NAME: TRNIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *TRANSISTOR card sequence according to the value of TYPE:

TYPE = 1 End of card sequence

TYPE = 2 Parameter card

TYPE = 6 Node card

VARIABLES:

ISW = End of data switch; Off = Ø, On = 1

NW = Word counter for parameter input

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

DATOUT DECIM ERROR NODEIN

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine TRNIN is the input processor for TRANSISTOR card sequences. Its function is to decode the bipolar junction

transistor input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine TRNIN causes a single input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine TRNIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

Subroutine DATOUT is called to complete the driver record and close the transistor definition. The input is tested for errors and the data and driver records are written to disk. The driver record is written to file 21 from the first ten words of global common, where:

MODE=9	Transistor Identifier
STADD	Data File Record Number
LNGTH=102	Length of Data Record in Words
MISC(3)	Base Node of Connection

The other MISC driver parameters are not used.

The transistor data record, written to file 20 from the common data buffer BUFF, is 102 words in length. This record allocated sufficient disk storage for the transistor input values as well as all other data generated internally by its model for use by subsequent NCAP phases:

BUFF(1)-(20)	Input Values

BUFF(21)-(70)	Nonlinear Parameters and Coefficients
BUFF(71)-(102)	Admittance Submatrix

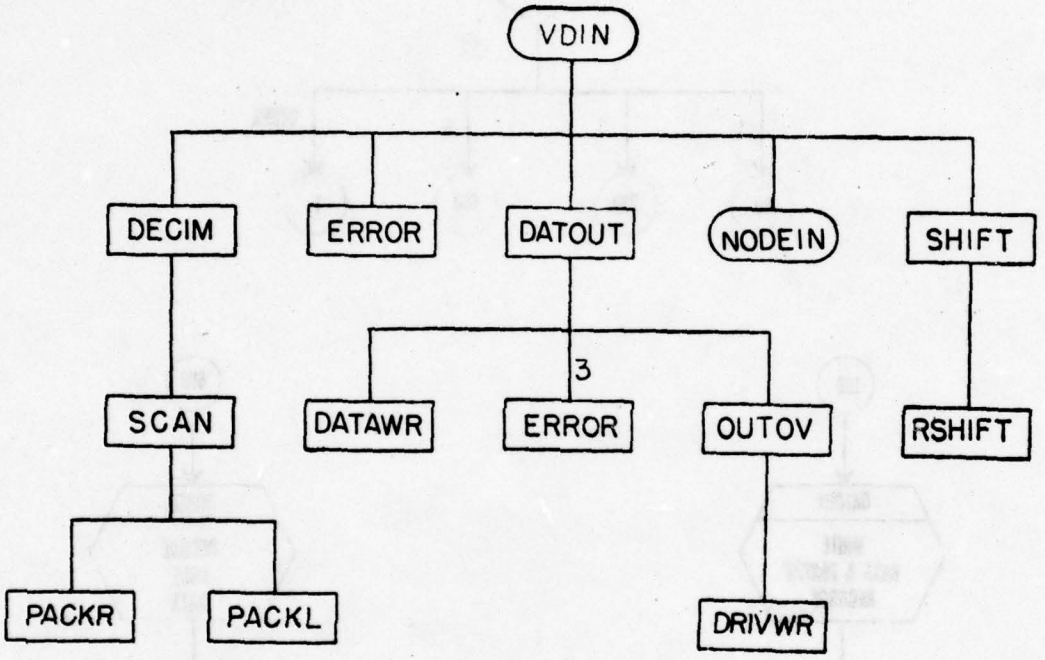
TYPE=2 Parameter Card

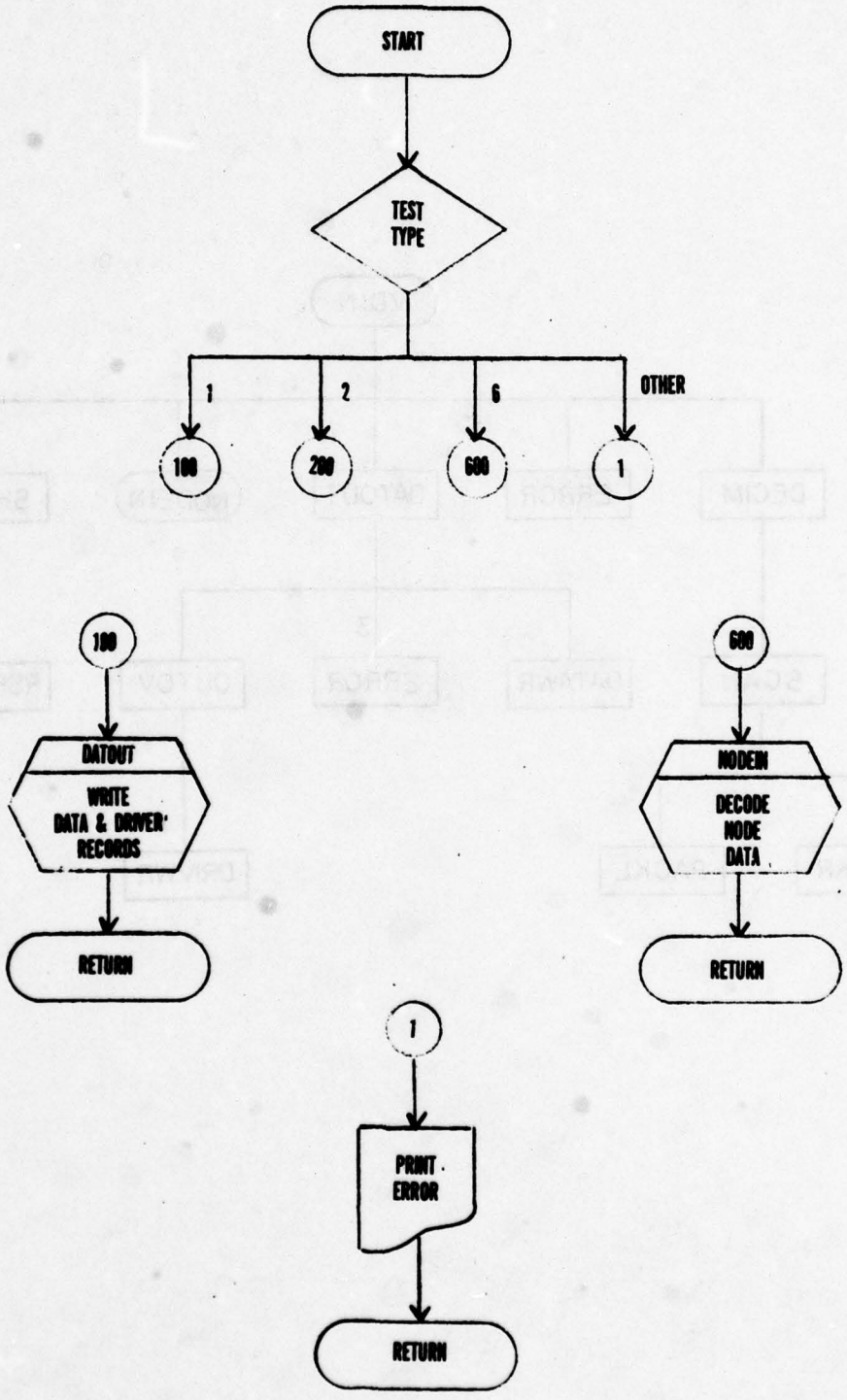
From one to sixteen decimal parameters are extracted from the card image, formatted, and stored in the data buffer by subroutine DECIM. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of values input on previous data cards. Parameter storage begins at BUFF(NW) where NW=NXT+1 and proceeds through successive buffer locations as data values are decoded by subroutine DECIM. If an end-of-data is encountered (ISW=1) before BUFF(16) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation on additional parameter cards.

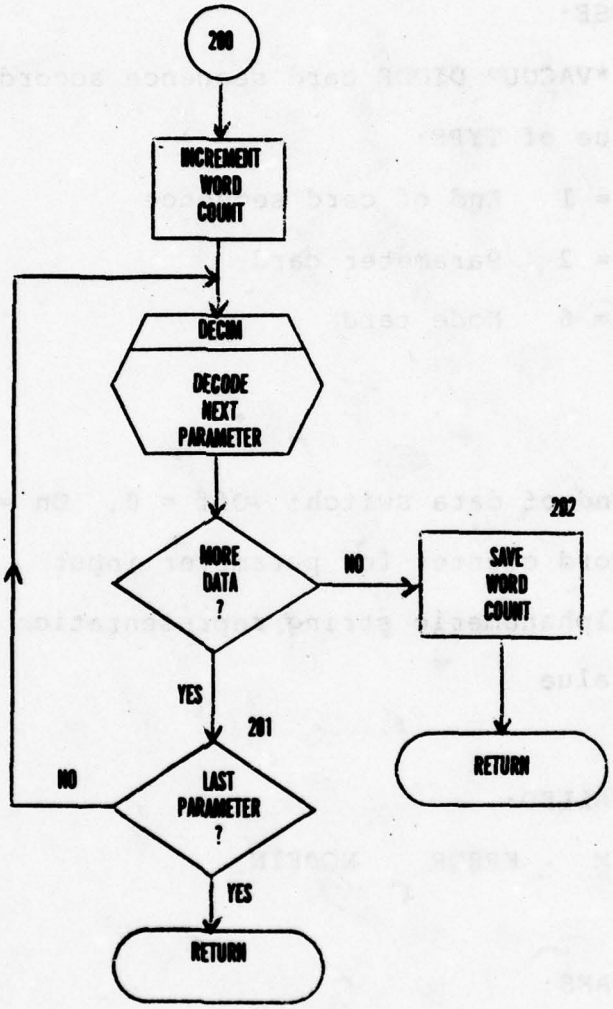
TYPE=6 Node Card

The base node of connection is extracted from the card image, formatted, and stored in MISC(3) by subroutine NODEIN.

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


VDIN
DECIM
ERROR
DATOUT
NODEIN
SHIFT
3
SCAN
DATAWR
ERROR
OUTOV
RSHIFT
PACKR
PACKL
DRIVWR


START
TEST TYPE
1
2
6
OTHER
100
200
600
1
100
DATOUT
WRITE DATA & DRIVER RECORDS
RETURN
600
MODEIN
DECODE MODE DATA
RETURN
1
PRINT ERROR
RETURN


200
INCREMENT WORD COUNT
DECOM
DECODE NEXT PARAMETER
MORE DATA ?
NO
202
SAVE WORD COUNT
RETURN
YES
201
LAST PARAMETER ?
NO
YES
RETURN

NAME: VDIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *VACUUM DIODE card sequence according to the value of TYPE.

TYPE = 1 End of card sequence

TYPE = 2 Parameter card

TYPE = 6 Node card

VARIABLES:

ISW = End of data switch; Off = 0, On = 1

NW = Word counter for parameter input

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

DATOUT DECIM ERROR NODEIN

CALLING PROGRAMS:

PPCSS

DESCRIPTION:

Subroutine VDIN is the input processor for VACUUM DIODE card sequences. Its function is to decode the vacuum diode input data

according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine VDIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine VDIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

Subroutine DATOUT is called to complete the driver record and to close the vacuum Diode definition. The input is tested for errors and the data and driver records are written to disk. The driver record is written to file 21 from the first ten words of global common, where:

MODE=6	Vacuum Diode Identifier
STADD	Data File Record Number
LNGTH=15	Length of Data Record in Words
MISC(3)	Positive Node of Connection
MISC(4)	Negative Node of Connection

Other MISC driver parameters are not used.

The vacuum diode data record, written to file 20 from the common data buffer BUFF, is 15 words in length. This record allocates sufficient disk storage for the vacuum diode input values as well as all other data generated internally by its model in subsequent NCAP phases:

BUFF(1)-(3)	Input Values
BUFF(4)-(13)	Nonlinear Coefficients
BUFF(14)-(15)	Admittance Submatrix

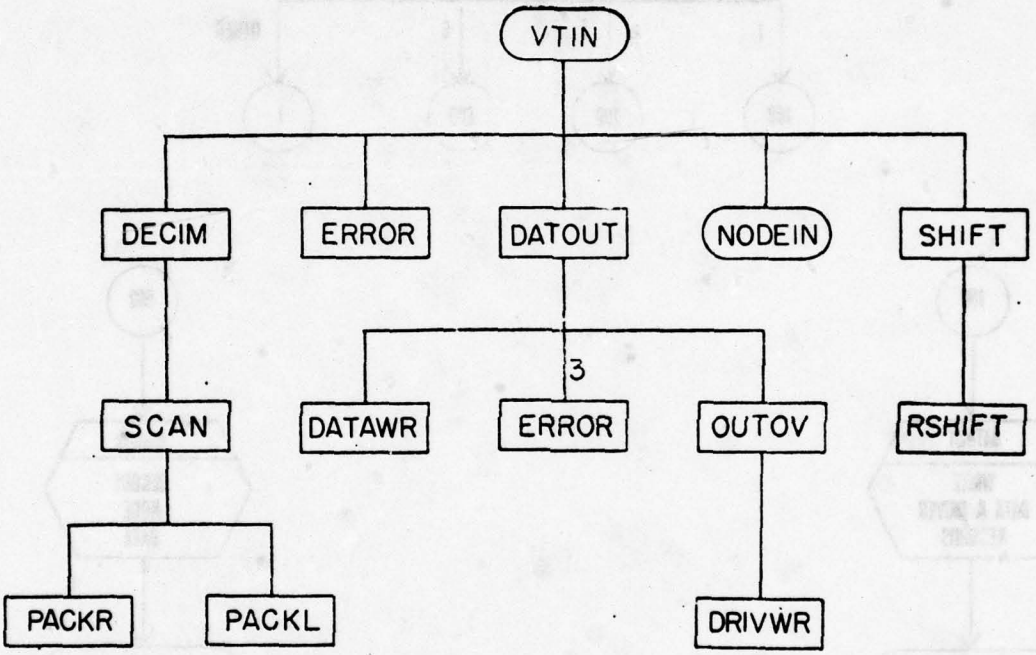
TYPE=2 Parameter Card

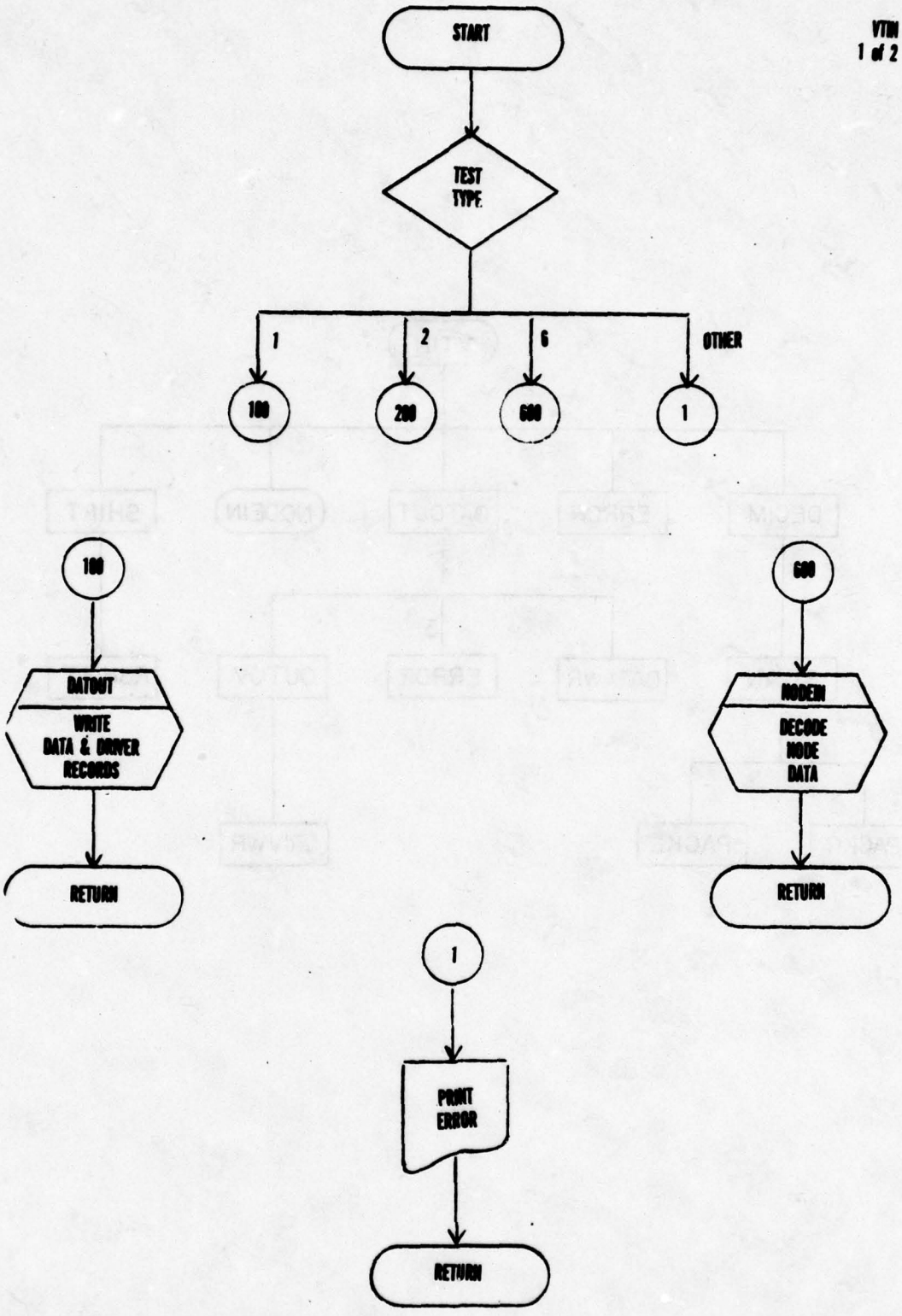
From one to three decimal parameters are extracted from the card image, formatted, and stored in the data buffer by subroutine DECIM. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of values input on previous data cards. Parameter storage begins at BUFF(NW) where NW=NXT+1 and proceeds through successive buffer locations as data values are decoded by subroutine DECIM. If an end-of-data is encountered(ISW=1) before BUFF(3) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation of additional parameter cards.

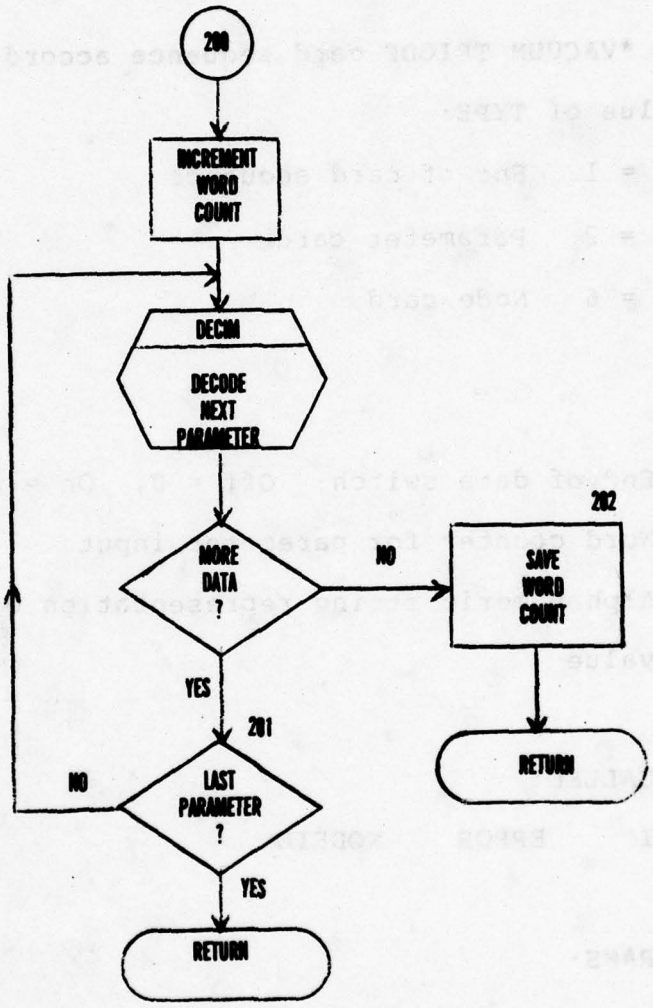
TYPE=6 Node Card

The nodes of connection are extracted from the card image, formatted, and stored in MISC(3) and MISC(4) by subroutine NODEIN.

Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.


VTIN
DECIM
ERROR
DATOUT
NODEIN
SHIFT
3
SCAN
DATAWR
ERROR
OUTOV
RSHIFT
PACKR
PACKL
DRIVWR


START
TEST TYPE
1
2
6
OTHER
100
200
600
1
100
DATOUT
WRITE DATA & DRIVER RECORDS
RETURN
600
MODEM
DECODE MODE DATA
RETURN
1
PRINT ERROR
RETURN


200
INCREMENT WORD COUNT
DECOM
DECODE NEXT PARAMETER
MORE DATA ?
NO
202
SAVE WORD COUNT
RETURN
YES
201
LAST PARAMETER ?
NO
YES
RETURN

NAME: VTIN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Processes *VACUUM TRIODE card sequence according to the value of TYPE.

- TYPE = 1 End of card sequence
- TYPE = 2 Parameter card
- TYPE = 6 Node card

VARIABLES:

ISW = End of data switch; Off = 0, On = 1

NW = Word counter for parameter input

STRING = Alphanumeric string representation of input value

SUBROUTINES CALLED:

DATOUT DECIM ERROR NODEIN

CALLING PROGRAMS:

PRCSS

DESCRIPTION:

Subroutine VTIN is the input processor for VACUUM TRIODE card sequences. Its function is to decode the vacuum triode

input data according to an anticipated format, translating the input into appropriate data and driver records for use by subsequent NCAP phases.

Each call to subroutine VTIN causes a single NCAP input card to be processed. The card image and its TYPE code are transmitted through common storage to subroutine VTIN which processes the card according to its TYPE as follows:

TYPE=1 End of Card Sequence

Subroutine DATOUT is called to complete the driver record and to close the Vacuum Triode definition. The input is tested for errors and the data and driver records are written to disk. The driver record is written to file 21 from the first ten words of global common, where

MODE=7	Vacuum Triode Identifier
STADD	Data File Record Number
LNGTH=40	Length of Data Record in Words
MISC(3)	Base Node of Connection

Other MISC driver parameters are not used.

The vacuum triode data record, written to file 20 from the common data buffer, BUFF, is 40 words in length. This record allocates sufficient disk storage for the vacuum triode input values as well as all other data generated internally by its model in subsequent NCAP phases:

BUFF(1)-(9)	Input Values

BUFF(10)-(22)	Nonlinear Parameters and Coefficients
BUFF(23)-(40)	Admittance Submatrix

TYPE=2 Parameter Card

From one to nine decimal parameters are extracted from the card image, formatted, and stored in the data buffer by subroutine DECIM. The number of values decoded and their placement in the buffer depend on the parameter count, NXT, and the end-of-data switch, ISW. Upon entering the subroutine, NXT contains the number of values input on previous data cards. Parameter storage begins at BUFF(NW) where NW=NXT+1 and proceeds through successive buffer locations as data values are decoded by subroutine DECIM. If an end-of-data is encountered (ISW=1) before BUFF(9) is filled, the parameter count is reset to the last buffer location used (NXT=NW) and control returns to the calling program in anticipation of additional parameter cards.

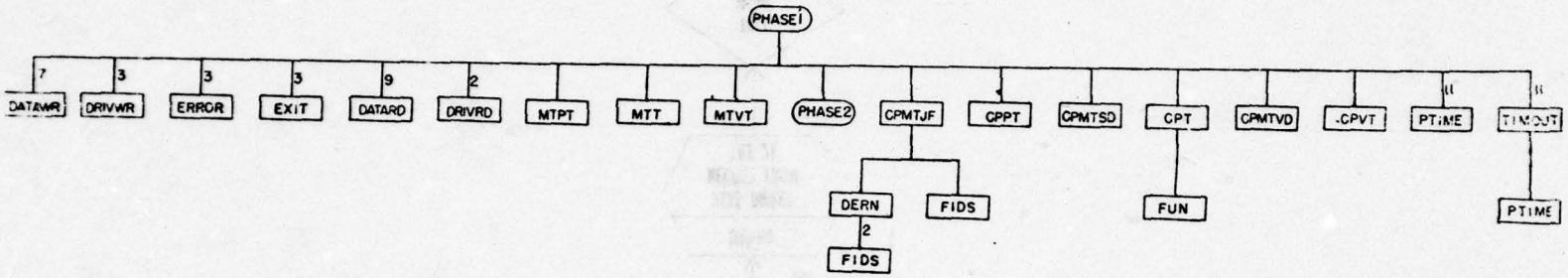
TYPE=6 Node Card

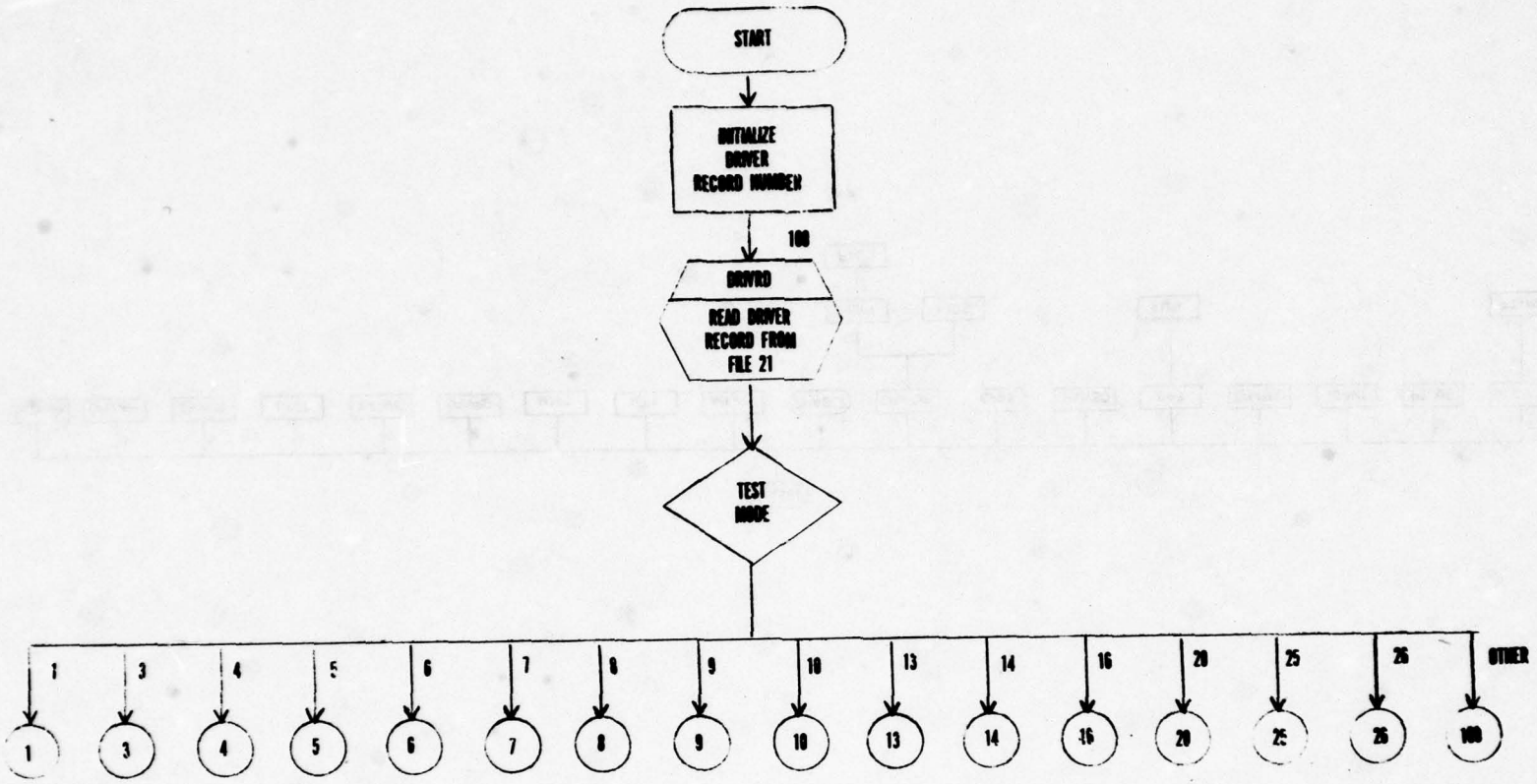
The base node of connection is extracted from the card image, formatted, and stored in MISC(3) by subroutine NODEIN.

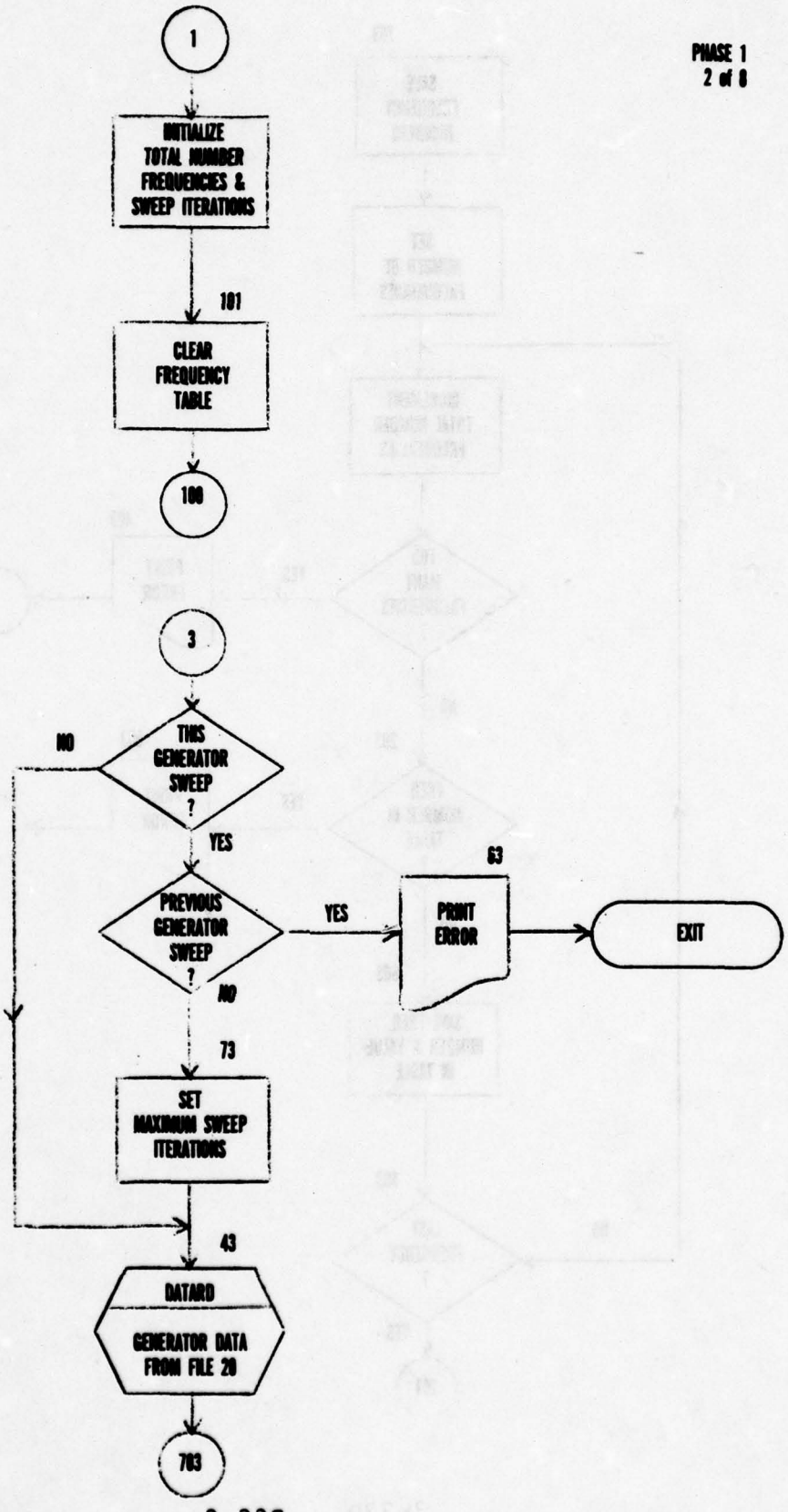
Any other value of TYPE constitutes an input error which is handled by subroutine ERROR before exit to the calling program.

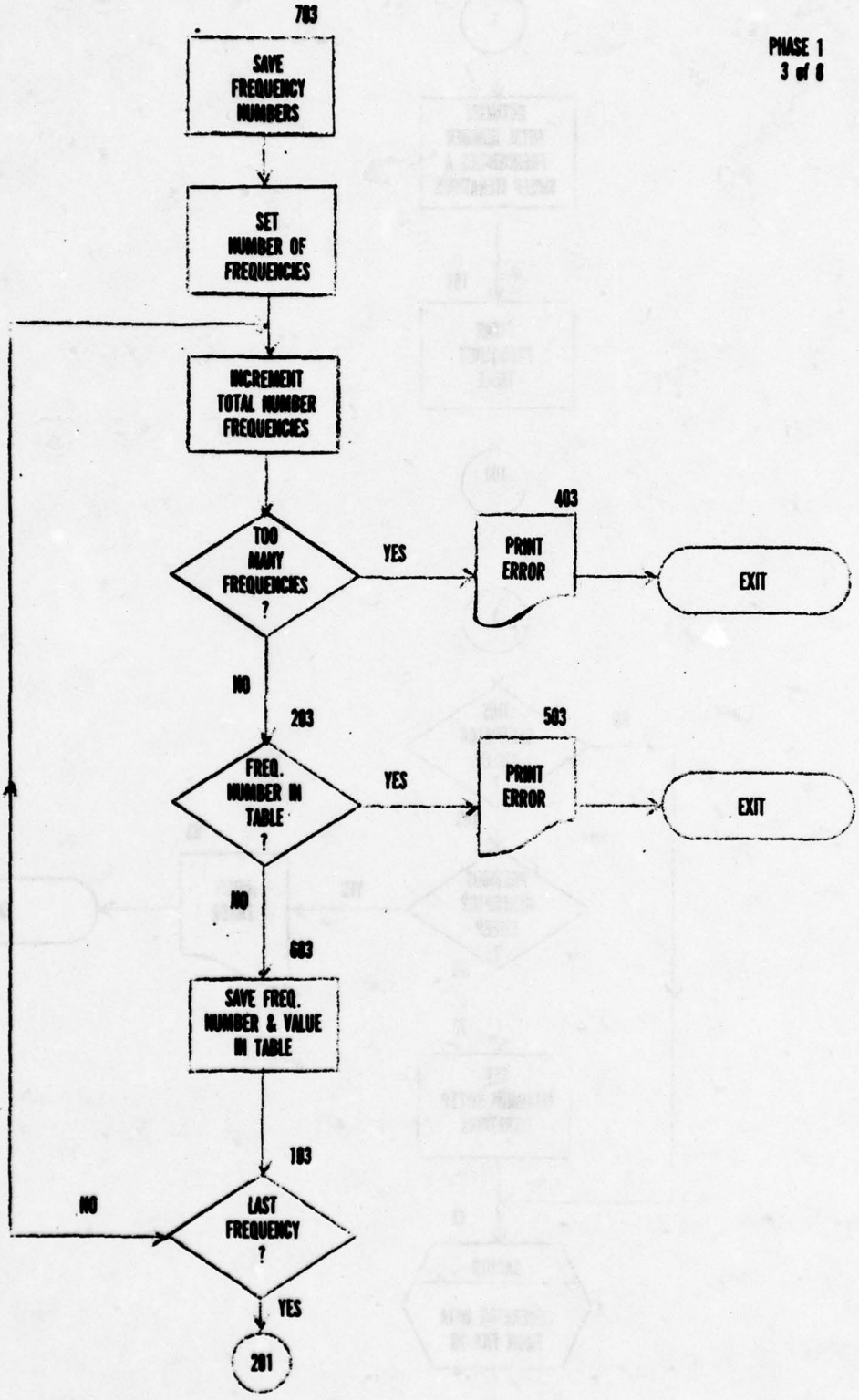
PHASE1

CPMTJF
CPMTSD
CPMTVD
CPPT
CPT
CPVT
DERIDS
FIDS
IMLOC
MTPT
MTT
MTVT

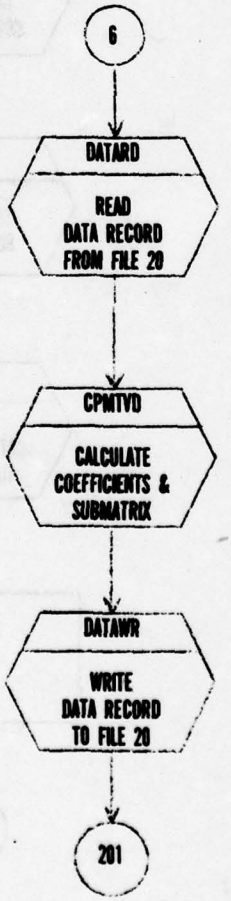

PHASE1
7
DATAWR
3
DRIVWR
3
ERROR
3
EXIT
9
DATARD
2
DRIVRD
MTPT
MTT
MTVT
PHASE2
CPMTJF
CPPT
CPMTSD
CPT
CPMTVD
CPVT
PTIME
TIMOUT
DERN
2
FIDS
FIDS
FUN
PTIME

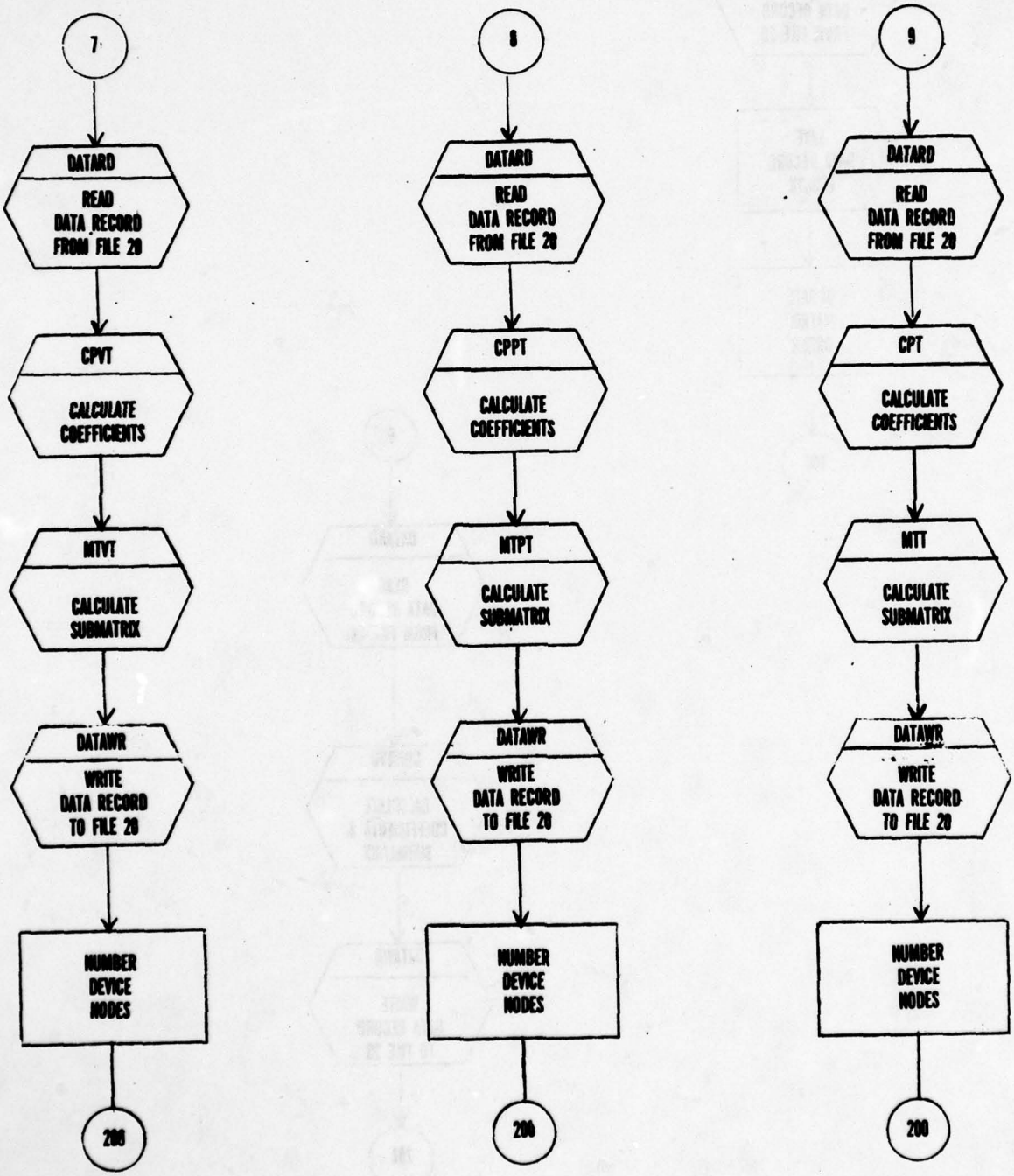

START
INITIALIZE
DRIVER
RECORD NUMBER
100
DRIVRD
READ DRIVER
RECORD FROM
FILE 21
TEST
MODE
1
3
4
5
6
7
8
9
10
13
14
16
20
25
26
OTHER
1
3
4
5
6
7
8
9
10
13
14
16
20
25
26
100

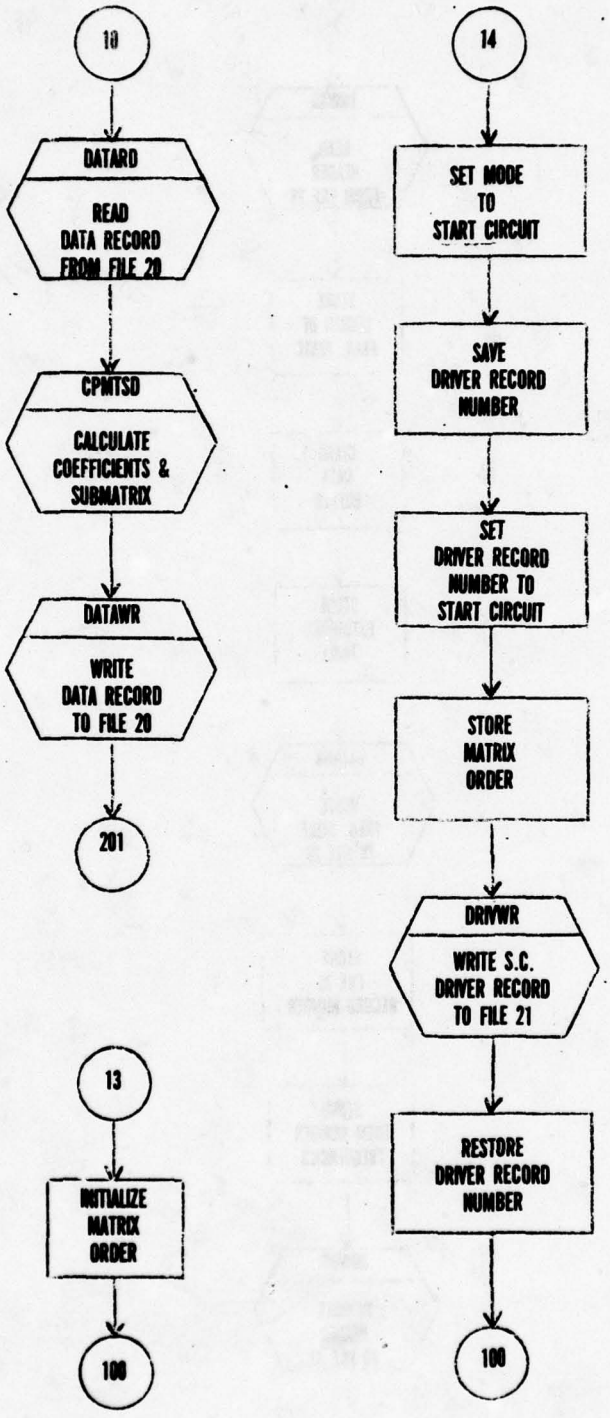

1
INITIALIZE TOTAL NUMBER FREQUENCIES & SWEEP ITERATIONS
101
CLEAR FREQUENCY TABLE
100
3
THIS GENERATOR SWEEP ?
NO
YES
PREVIOUS GENERATOR SWEEP ?
YES
63
PRINT ERROR
EXIT
NO
73
SET MAXIMUM SWEEP ITERATIONS
43
DATARD
GENERATOR DATA FROM FILE 20
703

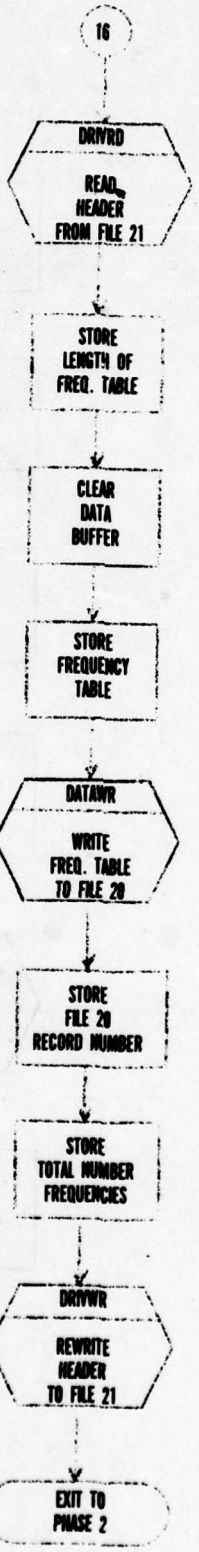

783
SAVE FREQUENCY NUMBERS
SET NUMBER OF FREQUENCIES
INCREMENT TOTAL NUMBER FREQUENCIES
403
TOO MANY FREQUENCIES ?
YES
PRINT ERROR
EXIT
NO
203
503
FREQ. NUMBER IN TABLE ?
YES
PRINT ERROR
EXIT
NO
603
SAVE FREQ. NUMBER & VALUE IN TABLE
103
NO
LAST FREQUENCY ?
YES
201

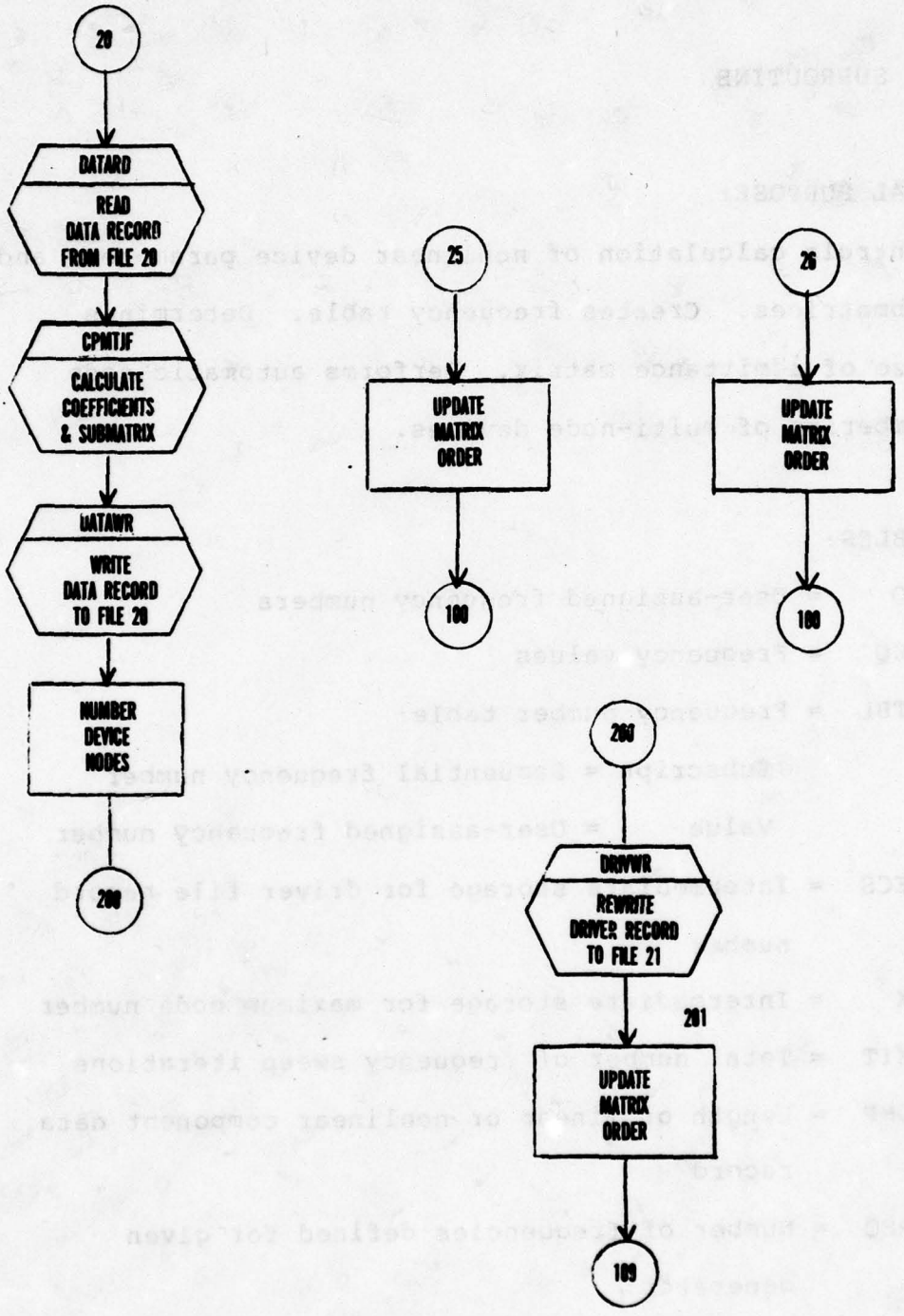

4
DATARD
READ
DATA RECORD
FROM FILE 20
SAVE
DATA RECORD
LENGTH
UPDATE
MATRIX
ORDER
100


6
DATARD
READ
DATA RECORD
FROM FILE 20
CPMTVD
CALCULATE
COEFFICIENTS &
SUBMATRIX
DATAWR
WRITE
DATA RECORD
TO FILE 20
201


7
DATARD
READ
DATA RECORD
FROM FILE 20
CPVT
CALCULATE
COEFFICIENTS
MTVT
CALCULATE
SUBMATRIX
DATAWR
WRITE
DATA RECORD
TO FILE 20
NUMBER
DEVICE
NODES
200
8
DATARD
READ
DATA RECORD
FROM FILE 20
CPPT
CALCULATE
COEFFICIENTS
MTPT
CALCULATE
SUBMATRIX
DATAWR
WRITE
DATA RECORD
TO FILE 20
NUMBER
DEVICE
NODES
200
9
DATARD
READ
DATA RECORD
FROM FILE 20
CPT
CALCULATE
COEFFICIENTS
MTT
CALCULATE
SUBMATRIX
DATAWR
WRITE
DATA RECORD
TO FILE 20
NUMBER
DEVICE
NODES
200


10
DATARD
READ
DATA RECORD
FROM FILE 20
CPMTSD
CALCULATE
COEFFICIENTS &
SUBMATRIX
DATAWR
WRITE
DATA RECORD
TO FILE 20
201
13
INITIALIZE
MATRIX
ORDER
100
14
SET MODE
TO
START CIRCUIT
SAVE
DRIVER RECORD
NUMBER
SET
DRIVER RECORD
NUMBER TO
START CIRCUIT
STORE
MATRIX
ORDER
DRIVWR
WRITE S.C.
DRIVER RECORD
TO FILE 21
RESTORE
DRIVER RECORD
NUMBER
100


16
DRIVRD
READ HEADER FROM FILE 21
STORE LENGTH OF FREQ. TABLE
CLEAR DATA BUFFER
STORE FREQUENCY TABLE
DATAWR
WRITE FREQ. TABLE TO FILE 20
STORE FILE 20 RECORD NUMBER
STORE TOTAL NUMBER FREQUENCIES
DRIVWR
REWRITE HEADER TO FILE 21
EXIT TO PHASE 2


20
DATARD
READ
DATA RECORD
FROM FILE 20
CPMTJF
CALCULATE
COEFFICIENTS
& SUBMATRIX
DATAWR
WRITE
DATA RECORD
TO FILE 20
NUMBER
DEVICE
NODES
200
25
UPDATE
MATRIX
ORDER
100
26
UPDATE
MATRIX
ORDER
100
200
DRVWR
REWRITE
DRIVER RECORD
TO FILE 21
201
UPDATE
MATRIX
ORDER
109

NAME: PHASE1

TYPE: SUBROUTINE

GENERAL PURPOSE:

Controls calculation of nonlinear device parameters and submatrices. Creates frequency table. Determines size of admittance matrix. Performs automatic node numbering of multi-node devices.

VARIABLES:

FNO = User-assigned frequency numbers

FREQ = Frequency values

FRTBL = Frequency number table:

Subscript = Sequential frequency number

Value = User-assigned frequency number

JRECS = Intermediate storage for driver file record number

MAX = Intermediate storage for maximum node number

MAXIT = Total number of frequency sweep iterations

NCOMP = Length of linear or nonlinear component data record

NFREQ = Number of frequencies defined for given generator

NODE = Maximum node number (order of admittance matrix)

NORDR = Total number defined frequencies (order of analysis)

SUBROUTINES CALLED:

CPMTJF	CPMTSD	CPMTVD	CPPT	CPT
CPVT	DATARD	DATAWR	DRIVRD	DRIVWR
FRROR	MTPT	MTT	MTVT	
PHASE2	PTIME	TIMOUT		

CALLING PROGRAMS:

ENDIN	PHASE4	PHASE5	PHASE6	PHASE7

DESCRIPTION:

Subroutine PHASE1 is the principal routine on NCAP's first computational phase. It creates the frequency table, controls the calculation of nonlinear device parameters and admittance submatrices, performs automatic node numbering of multi-node circuit elements, and determines the order of the admittance matrix.

The path of execution through subroutine PHASE1 is governed by a sequential scan of the driver file. The scan is initialized by setting the driver record number to the start of the file, DRVEC=1. Subroutine DRIVRD reads a driver record from file 21 and places it in the first ten words of common storage. PHASE1 responds to the MODE of the driver record, causing it to be processed as follows:

MODE=1 Driver File Header

The frequency count is initialized to NORDR=0, the frequency number table is cleared by FRTBL(I)=0 for I=1, 2,...10, and the number of frequency sweep iterations is set to MAXIT=0. The next file 20 location available for data storage is defined by DATREC=MISC(1) and control returns to the driver file scan.

MODE=13 Start Circuit

The order of the admittance matrix, defined as the largest node number in the circuit, is initialized to NODE=1 and control returns to the driver file scan.

MODE=3 Generator

Processing of the generator driver record begins by determining if more than one generator employs frequency sweeping. If the present generator does not specify sweeping (MISC(6) =0), the test is by-passed. However, if the present generator employs sweeping (MAXIT>0) and a previous generator also specified sweeping (MAXIT≠0), an appropriate error message is printed and program execution terminates. Otherwise the number of frequency sweep iterations, MAXIT, is set from MISC(6)

of the generator driver record.

The generator data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The user-assigned frequency numbers are transferred from BUFF(30)-(40) to the integer vector FN0 and the number of frequencies defined for the generator, NFREQ, is set from MISC(1).

Next the frequency numbers and values are appended to the frequency table as follows: for each defined frequency I=1, NFREQ, the sequential frequency number is incremented by NORDR=NORDR+1. If the frequency table is full (NORDR.GT.6), an appropriate error message is printed and program execution terminates. The user-assigned frequency number is tested for uniqueness against those already in the table. If FN0(I)= FRTBL(J) for any J=1, 2,...(NORDR-1), then an error message is printed and execution stops. Otherwise the frequency number and value are appended to the table by:

```
FRTBL(NORDR)=FN0(I)
FREQ(NORDR)=BUFF(I)
```

After all the generator frequencies have been placed in the table, the order of the admittance matrix is updated to the maximum of (NODE, MISC(3), MISC(4)) and control returns to the driver file scan.

MODE=4 Linear Components

The linear components data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The length of the data record, NCOMP, is set from MISC(1). For each four-word component definition the nodes of connection are extracted from the data buffer and stored in K and L and the order of the admittance matrix is updated to the maximum of (NODE, K, L). Control then returns to the driver file scan.

MODE=5 Nonlinear Components

The nonlinear components data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The length of the data record, NCOMP, is set from MISC (1). For each 13-word component definition the nodes of connection are extracted from the data buffer and stored in K and L and the order of the admittance matrix is updated to the maximum of (NODE, K, L). Control then returns to the driver file scan.

MODE=6 Vacuum Diode

The vacuum diode data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Subroutine CPMTVD is called to calculate the nonlinear parameters and admittance submatrix which are appended to the data record. The data record is rewritten to file 20 by subroutine DATAWR, the

order of the admittance matrix is updated to the maximum of (NODE, MISC(3), MISC(4)), and control returns to the driver file scan.

MODE=7 Vacuum Triode

The vacuum triode data record is read from file 20 by subroutine DATAPD and stored in the common data buffer BUFF. Subroutine CPVT and MTVT are called to calculate the vacuum triode nonlinear parameters and admittance submatrix which are appended to the data record. The data record is rewritten to file 20 by subroutine DATAWR and the nodes of the device are numbered by:

MISC(4)=MISC(3)+2

The driver record is rewritten to file 21 by subroutine DRIVWR, the order of the admittance matrix is updated to the maximum of (NODE, MISC(3), MISC(4)), and control returns to the driver file scan.

MODE=8 Vacuum Pentode

The vacuum pentode data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Subroutine CPPT and MTPT are called to calculate the vacuum pentode nonlinear parameters and admittance submatrix which are

appended to the data record. The data record is rewritten to file 20 by subroutine DATAWR and the nodes of the device are numbered by:

MISC(4)=MISC(3)+3

The driver record is rewritten to file 21 by subroutine DRIVWR, the order of the admittance matrix is updated to the maximum of (NODE, MISC(3), MISC(4)), and control returns to the driver file scan.

MODE=9 Bipolar Junction Transistor

The transistor data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Subroutines CPT and MTT are called to calculate the transistor nonlinear parameters and admittance submatrix which are appended to the data record. The data record is rewritten to file 20 by subroutine DATAWR and the nodes of the device are numbered by:

MISC(4)=MISC(3)+3

The driver record is rewritten to file 21 by subroutine DRIVWR, the order of the admittance matrix is updated to the maximum of (NODE, MISC(3), MISC(4)), and control returns to the driver file scan.

MODE=10 Semiconductor Diode

The semiconductor diode data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Subroutine CPMTSD is called to calculate the nonlinear parameters and admittance submatrix which are appended to the data record. The data record is rewritten to file 20 by subroutine DATAWR, the order of the admittance matrix is updated to the maximum of (NODE, MISC(3), MISC(4)), and control returns to the driver file scan.

MODE=20 Junction Field Effect Transistor

The junction field effect transistor data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Subroutine CPMTJF is called to calculate the nonlinear parameters and admittance submatrix which are appended to the data record. The data record is rewritten to file 20 by subroutine DATAWR and the nodes of the device are numbered by:

MISC(4)=MISC(3)+3

The driver record is rewritten to file 21 by subroutine DRIVWR, the order of the admittance matrix is updated to the maximum of (NODE, MISC(3), MISC(4)), and control returns to the driver file scan.

MODE=25 Linear Dependent Source

The order of the admittance matrix is updated to the maximum of (NODE, MISC(2), MISC(3), MISC(4), MISC(5)) and control returns to the driver file scan.

MODE=26 Nonlinear Dependent Source

The order of the admittance matrix is updated to the maximum of (NODE, MISC(2), MISC(3), MISC(4), MISC(5)) and control returns to the driver file scan.

MODE=14 End Circuit

The driver file record number is saved by JRECS=DRVREC and reset to that of the start circuit driver record by DRVREC =MISC(1). An updated start circuit (MODE=13) driver record is created with the order of the admittance matrix, NODE, stored in MISC(4). The start circuit driver record is rewritten to file 21 by subroutine DRIVWR, the driver file record number is restored to DRVREC=JRECS, and control returns to the driver file scan.

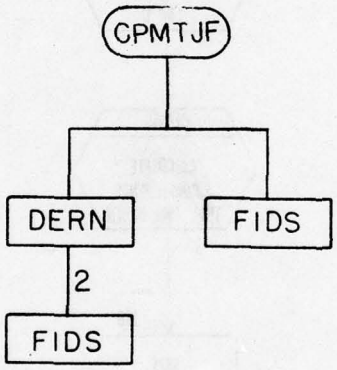
MODE=16 End

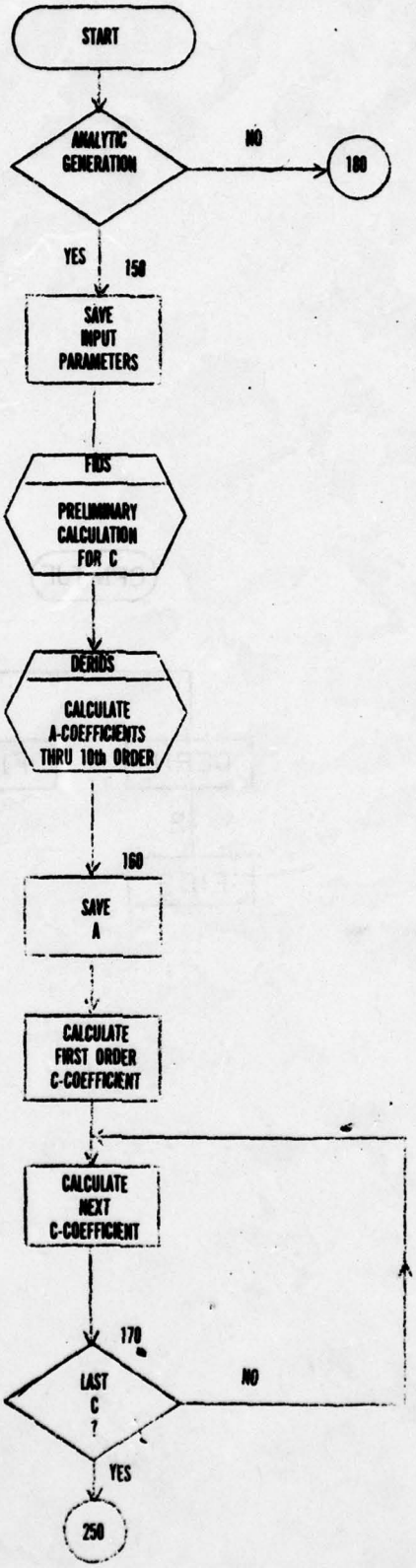
The driver file header is read from file 21 by subroutine DRIVRD. The frequency table is transferred from the FREQ and

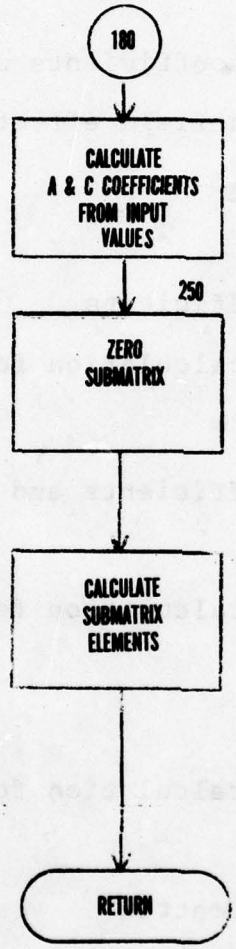
FRTBL arrays to the comon data buffer BUFF and output to file 20 by subroutine DATAWR. The record number and length of the frequency table are stored in the driver header record by STADD= DATREC and LNGTH=20. The next available file 20 storage location is updated by DATREC=DATREC+LNGTH and stored in MISC(1). The frequency count NORDR is stored in MISC(2) and the driver header is rewritten to file 21 by subroutine DRIVWR.

The FND driver record signals the end of the driver file scan and the completion of Phase 1 processing. Program control transfers to subroutine PHASE2.

For other values of MODE encountered in the driver file scan, control transfers back to the scan without further processing by subroutine PHASE1.


CPMTJF
DERN
FIDS
2
FIDS


START
ANALYTIC GENERATION
NO
100
YES
150
SAVE INPUT PARAMETERS
FDS
PRELIMINARY CALCULATION FOR C
DERDS
CALCULATE A-COEFFICIENTS THRU 10th ORDER
160
SAVE A
CALCULATE FIRST ORDER C-COEFFICIENT
CALCULATE NEXT C-COEFFICIENT
170
LAST C ?
NO
YES
250


100
CALCULATE
A & C COEFFICIENTS
FROM INPUT
VALUES
250
ZERO
SUBMATRIX
CALCULATE
SUBMATRIX
ELEMENTS
RETURN

NAME: CPMTJF

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear coefficients and creates admittance submatrix for junction field effect transistor

VARIABLES:

A = Nonlinear coefficients

AX = Intermediate calculation for A

B = Input values

C = Nonlinear coefficients and input values

CGD = Input value

D = Intermediate calculation for A

FK = Input value

FM = Input value

GIDS = Intermediate calculation for C

IDMAX = Input value

PJ = Admittance submatrix

RHO = Input Value

Q = Intermediate storage for input values

RS = Input value

VBI = Input value

VGS = Intermediate storage for input value

VGSI = Input value

VO = Input value

VP = Input value

VSUM = Intermediate calculation for C

W = Coefficient used in calculation for C

WM = Exponent used in calculation for C

ZFACT = Multiplicative accumulator for factorial

SUBROUTINES CALLED·

DERN FIDS

CALLING PROGRAMS:

PHASE1

DESCRIPTION:

Subroutine CPMTJF calculates the device parameters and admittance submatrix for the junction field effect transistor. The parameters are derived from the input values stored in the first 22 words of the JFET data record which is transmitted to subroutine CPMTJF in the common data buffer. The calculated device parameters and admittance submatrix are stored in the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

The FET model includes analytic generation of the nonlinear coefficients A and C from input values or the direct calculation of A and C coefficients from input parameters. The method of calculating coefficients is determined by the value of MISC(2) in the JFET driver record. For MISC(2)=2 the coefficients are

generated analytically; otherwise the coefficients are calculated directly.

The analytic generation of coefficients begins by storing the input B-parameters in the vector Q and saving B(5) in VGS. Then the function VGSI is calculated iteratively under control of the index I. VGSI is recursive in IDS, a function evaluated by subroutine FIDS. (Since the calculation of IDS is presented in detail in the narrative description of subroutine FIDS, it is not repeated here.) The algorithm for computing VGSI begins by evaluating IDS at the operating point X0=VGSI, an input parameter. Upon return from subroutine FIDS, the calculated value for IDS(X0) is stored in GIDS.

For successive iterations, the operating point is recomputed as X0 = VGSI - GIDS * RS and the function IDS(X0) is evaluated again. After the tenth iteration, the final result is added to V0 to form the expression:

$$VSUM = V0 + VGSI$$

The first order coefficient C(1) is then defined as:

$$C(1) = FK * |VSUM|^{-FM}$$

$$A(3) = A(3) + \frac{2. * B(2)^2 * RS}{D^5}$$

$$A(4) = A(4) + \frac{5. * B(2) * B(3) * RS}{D^6} + \frac{5. * B(2)^2 * RS^2}{D^7}$$

$$AX = \frac{(6. * B(2) * B(4) * RS) + (3. * B(3)^2 * RS)}{D^7}$$

$$A(5) = A(5) + AX$$

$$AX = \frac{21. * B(2)^2 * B(3) * RS^2}{D^8}$$

$$A(5) = A(5) + AX + \frac{14. * B(2)^4 * RS^3}{D^9}$$

The C-coefficients are calculated directly from the input C-vector according to:

$$C(I) = C(I) / I! \qquad \text{for } I = 2,3,\ldots 10$$

After the nonlinear coefficients have been calculated, subroutine CPMTJF calculates the complex admittances for the 4-node junction field effect transistor and stores them in the submatrix PJ. The submatrix storage area is first zeroed, and then the admittances are placed in the submatrix as follows:

Higher order coefficients C(I), I = 2,3,...10 are derived by differentiating C(I - 1) and dividing by (I - 1)! Since each C(I-1) is a function of the form mx^n, its derivative is computed directly as nmx^{n-1}. The coefficient of the derivative is developed in W, the exponent in WM, and the factorial expression is accumulated in ZFACT/I.

The analytic generation of A-coefficients is performed by subroutine DERIDS, which calculates the coefficients up to sixth order by differentiating the function IDS.

For the direct calculation of the FET nonlinear coefficients, the A-coefficients are derived from the input values RS and B. First the intermediate calculations to simplify expressions·

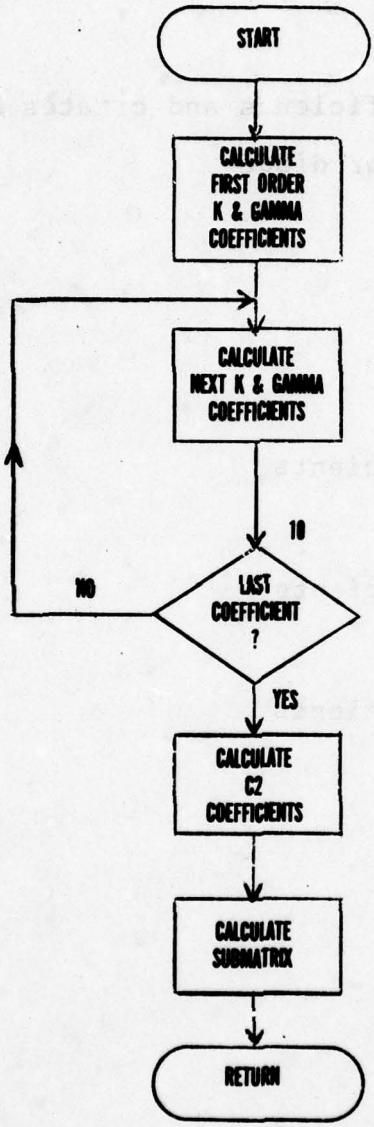
$$D = 1. - B(1) * RS$$

$$A(I) = B(I)/Z^{I+1} \quad \text{for } I = 2,3,...10$$

Then more specifically:

$$A(1) = B(1) / D$$

REAL	IMAG	REAL	IMAG	REAL	IMAG	REAL	IMAG
PJ(1,1,1)=	PJ(2,1,1)=	PJ(1,1,2)=	PJ(2,1,2)=	PJ(1,1,3)=	PJ(2,1,3)=	PJ(1,1,4)=	PJ(2,1,4)=
	C(1)						
0.	+CGD	0.	-CGD	0.	-C(1)	0.	0.
PJ(1,2,1)=	PJ(2,2,1)=	PJ(1,2,2)=	PJ(2,2,2)=	PJ(1,2,3)=	PJ(2,2,3)=	PJ(1,2,4)=	PJ(2,2,4)=
A(1)	-CGD	0.	CGD	-A(1)	0.	0.	0.
PJ(1,3,1)=	PJ(2,3,1)=	PJ(1,3,2)=	PJ(2,3,2)=	PJ(1,3,3)=	PJ(2,3,3)=	PJ(1,3,4)=	PJ(2,3,4)=
				A(1)+			
-A(1)	-C(1)	0.	0.	1./RS	C(1)	-1./RS	0.
PJ(1,4,1)=	PJ(2,4,1)=	PJ(1,4,2)=	PJ(2,4,2)=	PJ(1,4,3)=	PJ(2,4,3)=	PJ(1,4,4)=	PJ(2,4,4)=
0.	0.	0.	0.	-1./RS	0.	1./RS	0.


START
CALCULATE FIRST ORDER K & GAMMA COEFFICIENTS
CALCULATE NEXT K & GAMMA COEFFICIENTS
10
LAST COEFFICIENT ?
NO
YES
CALCULATE C2 COEFFICIENTS
CALCULATE SUBMATRIX
RETURN

NAME: CPMTSD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear coefficients and creates admittance submatrix for semiconductor diode

VARIABLES:

C	= Input value
CP	= Input value
C2	= Nonlinear coefficients
DNIF	= Input value
GAMMA	= Nonlinear coefficients
IFB	= Input value
K	= Nonlinear coefficients
MU	= Input value
P	= Submatrix
RRB	= Input value
SMK	= Input value
VRB	= Input value

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

PHASE1

DESCRIPTION:

Subroutine CPMTSD calculates the device parameters and admittance submatrix for the semiconductor diode. The parameters are derived from the input values stored in the first eight words of the semiconductor diode data record which is transmitted to subroutine CPMTSD in the common data buffer. The calculated device parameters and admittance submatrix are stored in the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

The semiconductor diode parameters include the coefficients K and C2 for the forward-biased model and the coefficients GAMMA for the reverse-biased model.

The calculation of coefficients K for the forward-bias model is recursive. The linear coefficient is given by:

K(1)= IFB/(.026 * DNIF)

Higher order coefficients are calculated according to:

K(I) = K(1) * K(I-1)/ (I * IFB)

The C2 parameters are defined as follows:

C2(1) = C + IFB * CP

C2(2) = CP

The calculation of coefficients GAMMA for the reverse-bias model is recursive. The linear coefficient is given by:

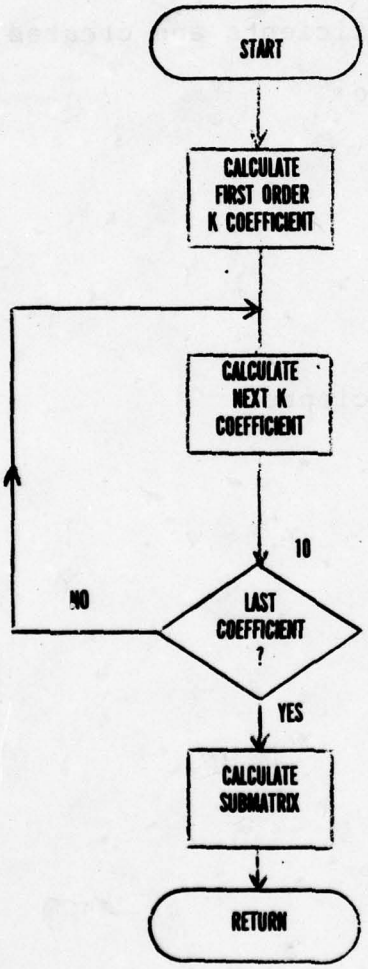
GAMMA(1) = SMK/VRB**MU

Higher order coefficients are calculated according to:

GAMMA(I) = (I-1-MU) * GAMMA(I-1)/(I*VRB)

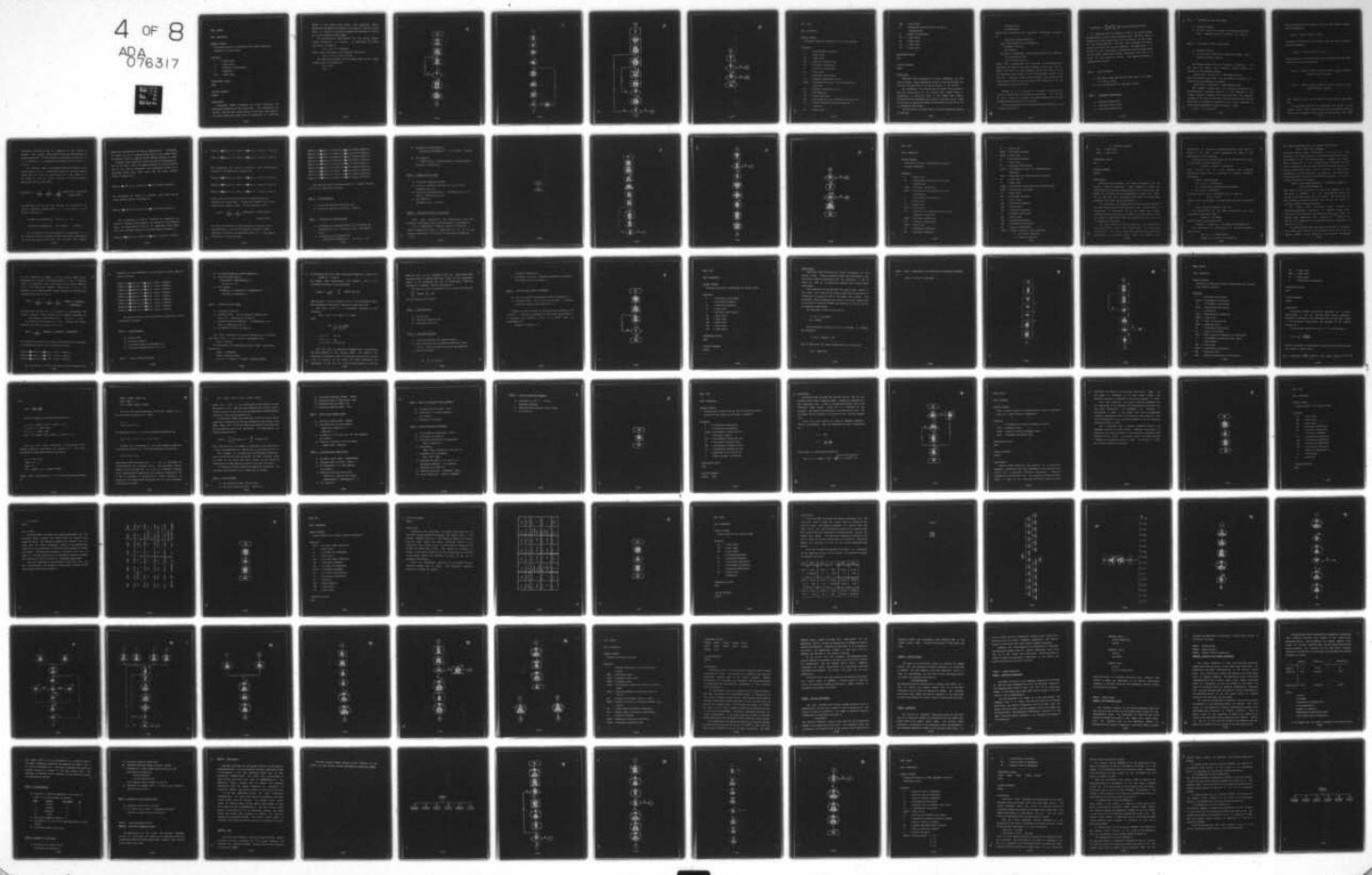
The admittance submatrix for the semiconductor diode is the single complex number represented by:

```
P(1) = K(1) + (1./RRB)
P(2) = C2(1) + GAMMA(1)
```


START
CALCULATE FIRST ORDER K COEFFICIENT
CALCULATE NEXT K COEFFICIENT
LAST COEFFICIENT ?
NO
YES
CALCULATE SUBMATRIX
RETURN

AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

4 OF 8
ADA 076317



NAME: CPMTVD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear coefficients and creates admittance submatrix for vacuum diode

VARIABLES:

C	= Input value
G	= Input value
K	= Nonlinear coefficients
P	= Submatrix
VPO	= Input value

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

PHASE1

DESCRIPTION:

Subroutine CPMTVD calculates the device parameters and admittance submatrix for the vacuum diode. The parameters are derived from the input values stored in the first three words of the vacuum diode data record which is transmitted to subroutine

CPMTVD in the common data buffer. The calculated device parameters and admittance submatrix are stored in the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

The calculation of coefficients K for the vacuum diode's single nonlinearity is recursive. By definition the linear coefficient is given by:

K(1) = 1.5 * G * SQRT(VPO)

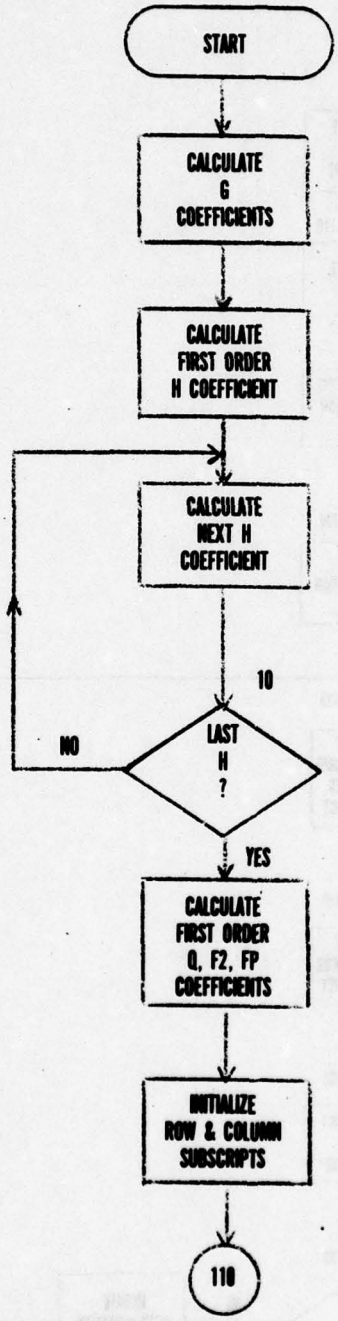
Higher order coefficients are calculated according to:

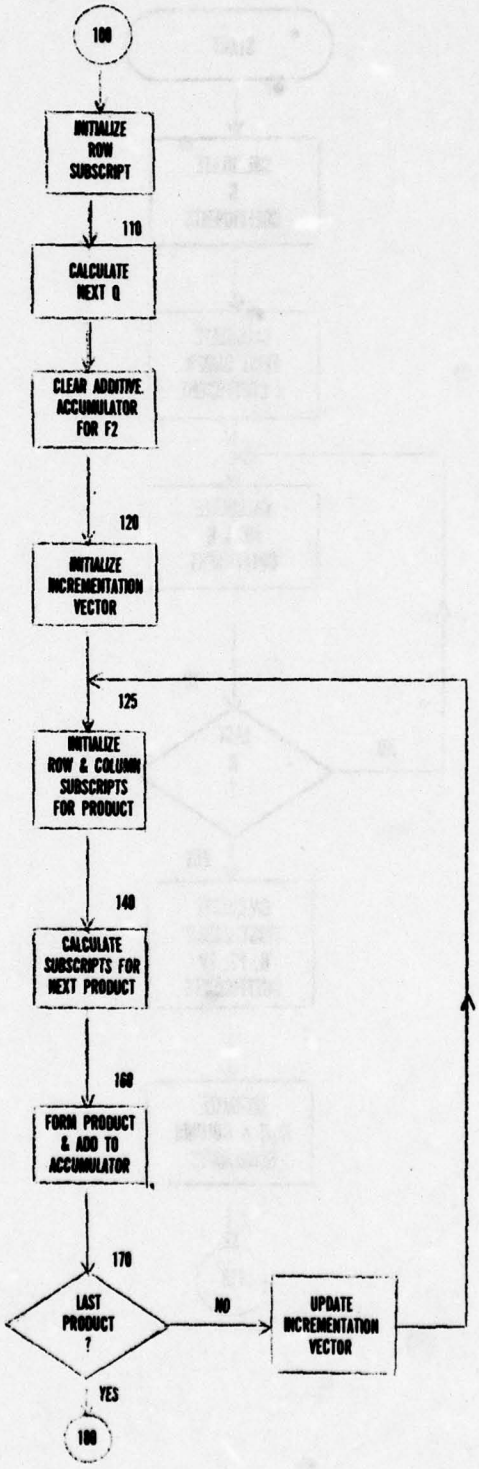
K(I) = (2.5-I) * K(I-1)/I*VPO

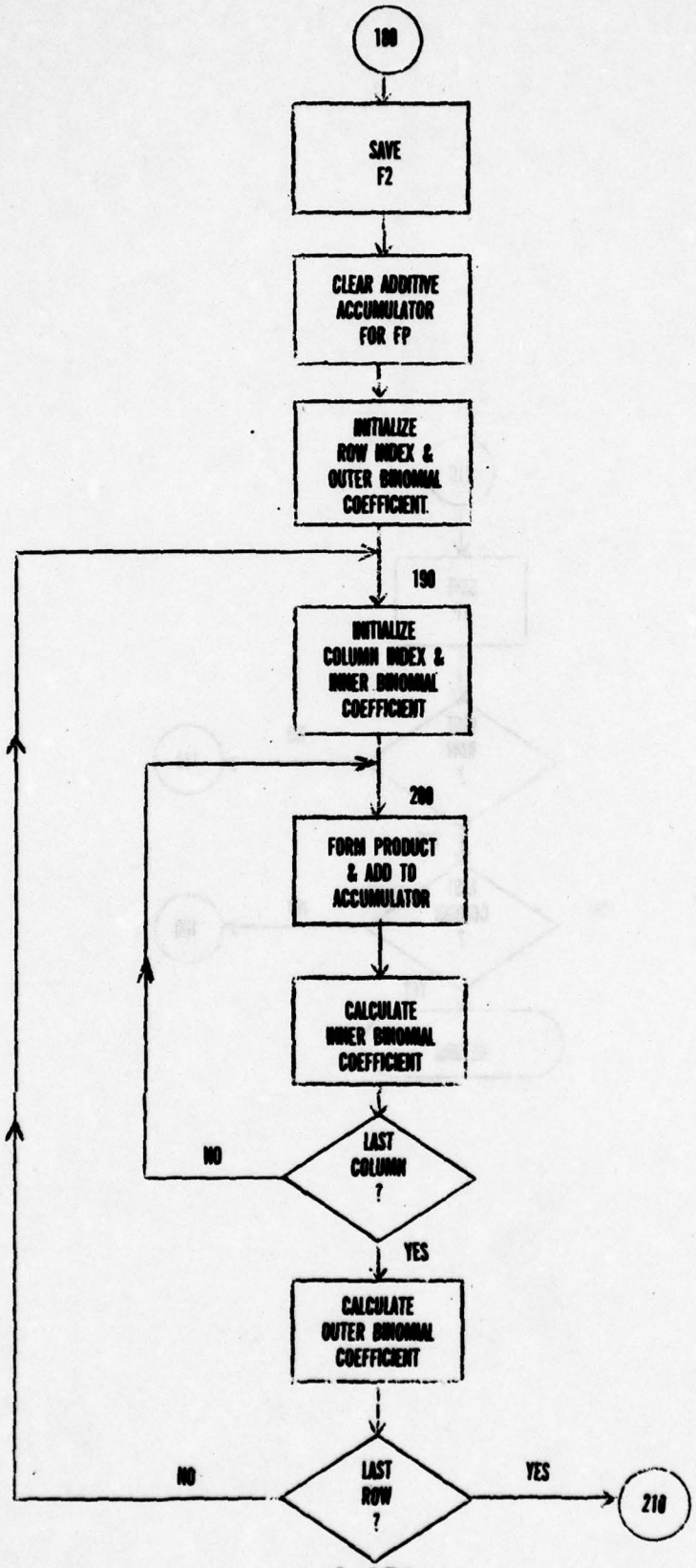
The admittance submatrix for the vacuum diode is the single complex number represented by:

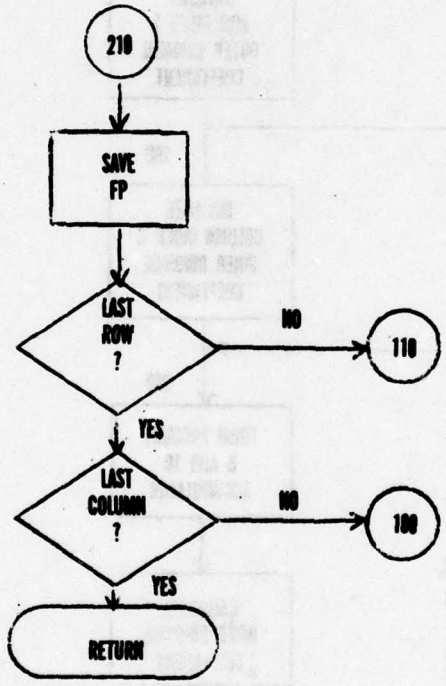
P(1) = K(1)

P(2) = C


START
CALCULATE G COEFFICIENTS
CALCULATE FIRST ORDER H COEFFICIENT
CALCULATE NEXT H COEFFICIENT
10
LAST H ?
NO
YES
CALCULATE FIRST ORDER Q, F2, FP COEFFICIENTS
INITIALIZE ROW & COLUMN SUBSCRIPTS
110


100
INITIALIZE ROW SUBSCRIPT
110
CALCULATE NEXT Q
CLEAR ADDITIVE ACCUMULATOR FOR F2
120
INITIALIZE INCREMENTATION VECTOR
125
INITIALIZE ROW & COLUMN SUBSCRIPTS FOR PRODUCT
140
CALCULATE SUBSCRIPTS FOR NEXT PRODUCT
160
FORM PRODUCT & ADD TO ACCUMULATOR
170
LAST PRODUCT ?
NO
UPDATE INCREMENTATION VECTOR
YES
180


100
SAVE F2
CLEAR ADDITIVE ACCUMULATOR FOR FP
INITIALIZE ROW INDEX & OUTER BINOMIAL COEFFICIENT
190
INITIALIZE COLUMN INDEX & INNER BINOMIAL COEFFICIENT
200
FORM PRODUCT & ADD TO ACCUMULATOR
CALCULATE INNER BINOMIAL COEFFICIENT
LAST COLUMN ?
NO
YES
CALCULATE OUTER BINOMIAL COEFFICIENT
LAST ROW ?
NO
YES
210


210
SAVE FP
LAST ROW ?
NO
110
YES
LAST COLUMN ?
NO
100
YES
RETURN

NAME: CPPT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear coefficients for vacuum pentode

VARIABLES:

A	= Intermediate calculation
D	= Input value
EM	= Input value
FP	= Nonlinear coefficients
F2	= Nonlinear coefficients
G	= Nonlinear coefficients
GO	= Input value
H	= Nonlinear coefficients
IC	= Binomial coefficient for JP
INDX	= Incrementation vector for differentiation algorithm
JC	= Binomial coefficient for J2
JP	= Row subscript
J2	= Column subscript
KP	= Row subscripts for differentiation algorithm
K2	= Column subscripts for differentiation algorithm
MU	= Input value

PHI = Input value

Q = Preliminary calculation for FP and F2 nonlinearities

S = Additive accumulator

VP = Input value

V1 = Input value

V1T = Input value

V2 = Input value

SUBROUTINES CALLED·

NONE

CALLING PROGRAMS:

PHASE1

DESCRIPTION·

Subroutine CPPT calculates the device parameters for the vacuum pentode. These parameters consist of the coefficients for the four nonlinearities of the device model G, H, F2, and FP.

The parameters are derived from the input values stored in the first twelve words of the pentode data record which is transmitted to subroutine CPPT in the common data buffer. Calculated device parameters are stored in the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

The calculaton of coefficients for the H nonlinearity begins by defining·

```
A=V1+PH1+V2/MU
H(1)=SQRT(A**3)
```

Higher order coefficients are calculated recursively according to:

```
H(I+1)=(2.5-FLOAT(I))*H(I)/A
```

The G coefficients are calculated by:

```
G(1)=GO*(1.-V1/V1T)
G(2)=-GO/V1T
```

For the F2 coefficients, the computations begin by defining:

```
Q(1,1)=D*(VP/V2)**EM
F2(1,1)=1./(1.+Q(1,1))
```

Higher order coefficients are generated by a differentiation algorithm and stored in the upper triangle of the array F2(5,5). Within the first row, each successive column entry F2(1,J2+1) is the derivative with respect to V2 of the preceding column entry F2(1,J2). Similarly, within a given column, each successive row entry F2(JP+1, J2+1) is the derivative with respect to VP of the preceding row entry F2(JP,J2+1).

Because F2 is a function of Q, an upper triangular array Q(5,5) is constructed to simplify the arithmetic. The elements of the Q array are derived and stored according to the same differentiation process as the F2 array.

The FP coefficients are calculated directly from the F2 and Q arrays according to:

$$FP(JP+1,J2+1) = \sum_{I=0}^{JP} \binom{JP}{I} \sum_{J=0}^{J2} \binom{J2}{J} F_2(JP-I+1,J2-J+1)\, Q(I+1,J+1)$$

In subroutine CPPT the elements of the Q, F2, and FP arrays are generated by columns within a double loop. The outer loop is indexed by J2 which ranges from Ø to 4 and serves as the basis for the column subscript. The inner loop is indexed by JP which serves as the basis for the row subscript. JP ranges from 1 to (4-J2) for the first column, effectively omitting calculations for the linear coefficients already evaluated, and from Ø to (4-J2) for the remaining columns. The indexing technique is summarized as follows:

STEP 1: Initialization

a) Set column index J2=Ø and row index JP=1. Go to Step 2 for first column.

b) Set row index JP=Ø for successive columns.

STEP 2: Calculate Coefficients

a) Calculate Q(JP+1,J2+1)

b) Calculate F2(JP+1,J2+1)

c) Calculate FP(JP+1,J2+1)

STEP 3: Increment and Test Row Index

a) Increment JP=JP+1

b) If JP.LE. (4-J2) go to Step 2 to calculate next row entry. Otherwise column is complete; go to Step 4.

STEP 4: Increment and Test Column Index

a) Increment J2=J2+1

b) If J2.LE.4 go to Step 1b to begin next column. Otherwise calculations complete.

The elements Q(JP+1,J2+1) are calculated recursively. For the first row (JP=Ø), each successive column element is a function of the preceding element:

Q(JP+1,J2+1) = Q(JP+1,J2) * (-EM-FLOAT(J2-1))/V2

For the remaining rows (JP>Ø), each successive row element is a function of the preceding row element:

Q(JP+1,J2+1) = Q(JP,J2+1) * (EM-FLOAT(JP-1))/VP

The elements F2(JP+1,J2+1) are calculated according to a differentiation algorithm similiar to that employed in the bipolar junction transistor (see subroutine CPT). The algorithm is based on the rule of elementary calculus which defines the nth derivative of a product of functions to be a sum of products of lower order derivatives of those functions. Given:

$$F_2(1,1) = [\ 1 + Q(1,1)\ ]^{-1}$$

then its derivative with respect to V2, the next column element, can be expressed as:

$$F_2(1,2) = -Q(1,2) * F_2(1,1) * F_2(1,1)$$

Similarly the derivative with respect to VP, the next row element can be expressed as:

$$F_2(2,1) = -Q(2,1) * F_2(1,1) * F_2(1,1)$$

These products of functions yield sums of products of lower order derivatives when differentiated:

$$F_2(1,3) = [Q(1,3) * F_2(1,1) * F_2(1,1)] + [Q(1,2) * F_2(1,2) * F_2(1,1)] + [Q(1,2) * F_2(1,1) * F_2(1,2)]$$

$$F_2(3,1) = [Q(3,1) * F_2(1,1) * F_2(1,1)] + [Q(2,1) * F_2(2,1) * F_2(1,1)] + [Q(2,1) * F_2(1,1) * F_2(2,1)]$$

Each additive term is the product of one Q and two lower-order F2's.

In subroutine CPPT the row subscripts are stored in the integer vector KP(3), where KP(1) is the row subscript of the Q element of the product, KP(2) is the row subscript of the first

F2 element, and KP(3) is the row subscript of the second F2 element. The integer vector K2(3) serves the same purpose for column subscripts. In this context the subscript vectors KP= (1, 1, 1) and K2=(3, 1, 1) represents the product Q(1,3) * F2(1,1) * F2(1,1).

The additive terms are generated under the control of the vector INDX(L), L=1, 2...(J2+JP) where INDX(1)=1 and each INDX(L) ranges from 1 to 3 for all other values of L. The elements of the INDX, KP, and K2 vectors interact to form the complete expression for each F2 element:

$$F_2(JP+1,J2+1) = \sum_{INDX(2)=1}^{3} \sum_{INDX(3)=1}^{3} \cdots \sum_{INDX(JP+J2)=1}^{3} Q(KP(1),K2(1) * F_2(KP(2),K2(2 * F_2(KP(3),K2(3))$$

The subscripts for each term are derived by incrementing the initial subscript vectors KP=(1, 1, 1) and K2=(1, 1, 1) by INDX(L) according to:

KP(INDX(L))=KP(INDX(L))+1 for L=1, 2 ... J2

K2(INDX(L))=K2(INDX(L))+1 for L=(J2+1), ... (J2+JP)

It is important to note that INDX(1) is always equal to 1, and that the first J2 elements of INDX define the incrementation of the column subscripts K2 while the remaining INDX elements

define the incrementation of the row subscripts KP. Furthermore for given JP and J2 (where F2(JP+1,J2+1) is the element being calculated), there are exactly (JP+J2) defined elements in INDX.

Together these constraints force the correct formulation for F2(1,2) and F2(2,1) which are simply products of the form Q*F2*F2 and do not involve summations. The formulation of F2(1,2) is summarized below where JP=Ø, J2=1, and the single element INDX=(1) increments K2:

INDX=(1) ➡ KP=(1,1,1), K2=(2,1,1) ➡ Q(1,2)*F2(1,1)*F2(1,1)

The calculation for F2(2,1) is similar. JP=1, J2=Ø, and the single element INDX=(1) increments KP:

INDX=(1) ➡ KP=(2,1,1), K2=(1,1,1) ➡ Q(2,1)*F2(1,1)*F2(1,1)

The calculation of F2(1,3) involves the summation for INDX(2)=1 through 3 with INDX(1)=1 and results in three additive terms. The formulation of F2(1,3) is summarized below where JP=Ø, J2=2, and both elements of the INDX vector increment K2:

INDX=(1,1) ➡ KP=(1,1,1), K2=(3,1,1) ➡ Q(1,3) * F2(1,1) * F2(1,1)

INDX=(1,2) ➡ KP=(1,1,1), K2=(2,2,1) ➡ Q(1,2) * F2(1,2) * F2(1,1)

INDX=(1,3) ➡ KP=(1,1,1), K2=(2,1,2) ➡ Q(1,2) * F2(1,1) * F2(1,2)

The calculation for F2(3,1) is similiar. JP=2, J2=0 and both elements of the INDX vector increment KP:

INDX=(1,1) ➡ KP=(3,1,1), K2=(1,1,1) ➡ Q(3,1) * F2(1,1) * F2(1,1)

INDX=(1,2) ➡ KP=(2,2,1), K2=(1,1,1) ➡ Q(2,1) * F2(2,1) * F2(1,1)

INDX=(1,3) ➡ KP=(2,1,2), K2=(1,1,1) ➡ Q(2,1) * F2(1,1) * F2(2,1)

Higher order derivatives involve more summations and are therefore more complicated. Consider for example F2(2,3) which contains nine additive terms. In this case JP=1, J2=2.

$$F_2(2,3) = -\sum_{INDX(2)=1}^{3} \sum_{INDX(3)=1}^{3} Q(KP(1),K2(1)) * F_2(KP(2),K2(2)) * F_2(KP(3),K2(3))$$

The first J2=2 elements of INDX increment the initial column subscripts K2=(1,1,1) and the remaining 1 element of INDX increments the initial row subscripts KP=(1,1,1). The complete formulation is tabulated below.

INDX=(1,1,1) ➡ KP=(2,1,1),K2=(3,1,1) ➡ Q(2,3)*F2(1,1)*F2(1,1)

INDX=(1,1,2) ➡ KP=(1,2,1),K2=(3,1,1) ➡ Q(1,3)*F2(2,1)*F2(1,1)

INDX=(1,1,3) ➡ KP=(1,1,2),K2=(3,1,1) ➡ Q(1,3)*F2(1,1)*F2(2,1)

INDX=(1,2,1) ➡ KP=(2,1,1),K2=(2,2,1) ➡ Q(2,2)*F2(1,2)*F2(1,1)

INDX=(1,2,2) ➡ KP=(1,2,1),K2=(2,2,1) ➡ Q(1,2)*F2(2,2)*F2(1,1)

INDX=(1,2,3) ➡ KP=(1,1,2),K2=(2,2,1) ➡ Q(1,2)*F2(1,2)*F2(2,1)

INDX=(1,3,1) ➡ KP=(2,1,1),K2=(2,1,2) ➡ Q(2,2)*F2(1,1)*F2(1,2)

INDX=(1,3,2) ➡ KP=(1,2,1),K2=(2,1,2) ➡ Q(1,2)*F2(2,1)*F2(1,2)

INDX=(1,3,3) ➡ KP=(1,1,2),K2=(2,1,2) ➡ Q(1,2)*F2(1,1)*F2(2,2)

The algorithm used in subroutine CPPT to compute F2(JP+1, J2+1) can be summarized as follows:

STEP 1· Initialization

a) Initialize additive accumulator S=Ø.

b) Initialize INDX(I)=1 for I=1,2,...(JP+J2)

STEP 2· Calculate One Additive Term

a) Initialize row subscripts KP=(1,1,1) and column subscripts K2=(1,1,1). If J2=Ø go to 2C. Otherwise go to 2b.

b) Increment column subscripts by:

K2(INDX(L))=K2(INDX(L))+1 for L=1,2,...,J2

c) Increment row subscripts by:

KP(INDX(L))=KP(INDX(L))+1 for L=(J2+1),...(JP+J2)

d) Form product:

Q(KP(1),K2(1)) * F2(KP(2),K2(2)) * F2(KP(3),K2(3))

and add to accumulator S.

STEP 3: Update and Test INDX

a) Initialize subscript T=JP+J2.

b) If T.LE.1, summation complete in S, go to Step 4. Otherwise go to 3c.

c) Increment INDX(T)=INDX(T)+1. If INDX(T).LE.3, go to Step 2. Otherwise go to 3d.

d) Set INDX(T)=1.

Decrement T=T-1. Go to 3b.

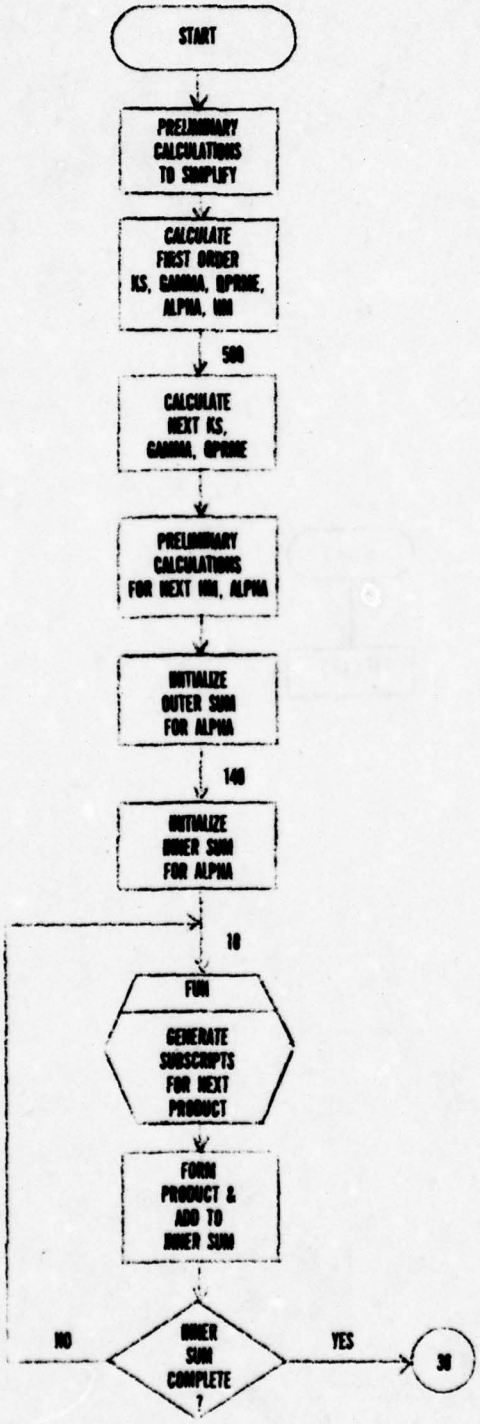
STEP 4: Set F2(JP+1,J2+1) = -S and Exit

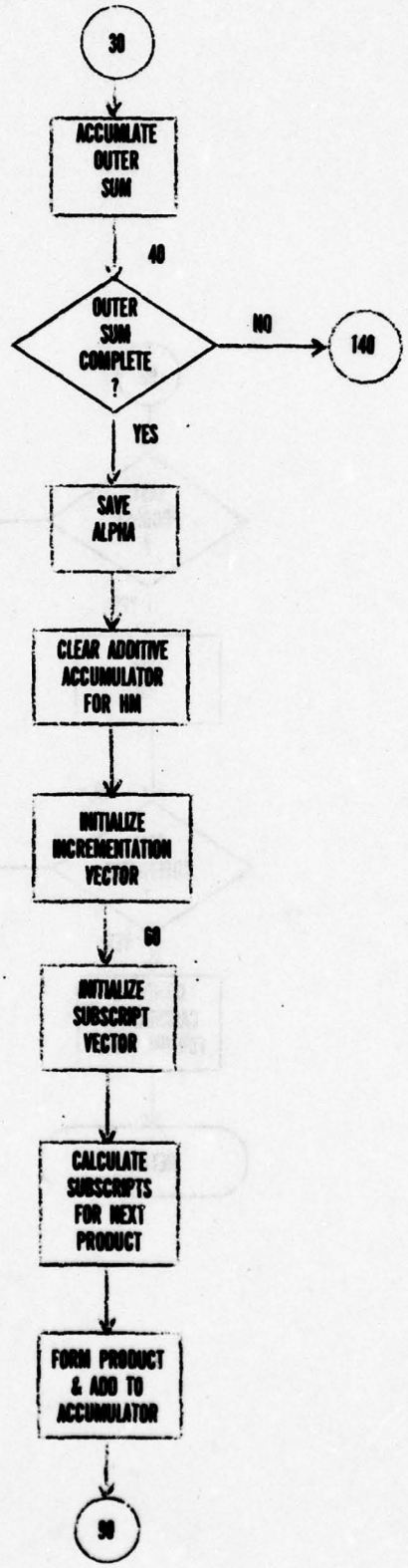
After each Q(JP+1,J2+1) and F2(JP+1,J2+1) have been computed, then FP(JP+1,J2+1) is accumulated in S according to:

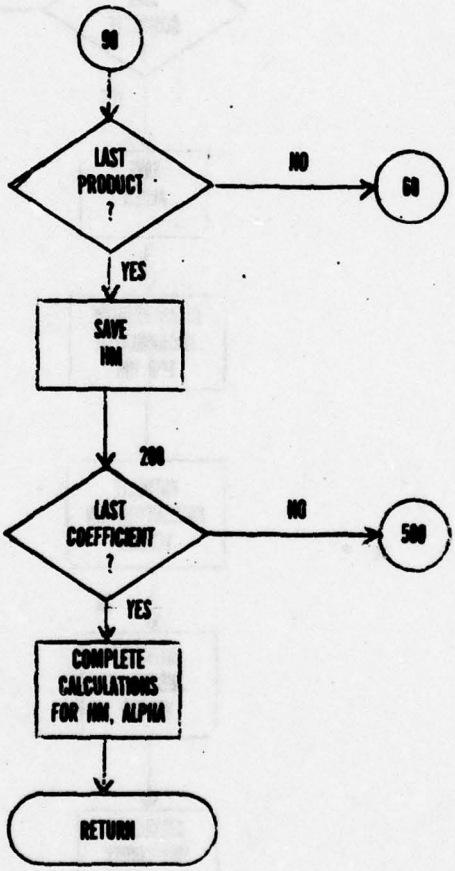
S = S + FLOAT(IC*JC) * F2(JP-I+1,J2-J+1) * Q(I+1,J+1)

where I ranges from 0 to JP, J ranges from 0 to J2, IC is the binomial coefficient $\left[\begin{smallmatrix} JP \\ I \end{smallmatrix} \right]$ and JC is the binomial coefficient $\left[\begin{smallmatrix} J2 \\ J \end{smallmatrix} \right]$.


CPT
FUN


START
PRELIMINARY CALCULATIONS TO SIMPLIFY
CALCULATE FIRST ORDER KS, GAMMA, QPRIME, ALPHA, MM
500
CALCULATE NEXT KS, GAMMA, QPRIME
PRELIMINARY CALCULATIONS FOR NEXT MM, ALPHA
INITIALIZE OUTER SUM FOR ALPHA
140
INITIALIZE INNER SUM FOR ALPHA
10
FOR
GENERATE SUBSCRIPTS FOR NEXT PRODUCT
FORM PRODUCT & ADD TO INNER SUM
INNER SUM COMPLETE ?
NO
YES
30


30
ACCUMULATE OUTER SUM
40
OUTER SUM COMPLETE ?
NO
140
YES
SAVE ALPHA
CLEAR ADDITIVE ACCUMULATOR FOR HM
INITIALIZE INCREMENTATION VECTOR
60
INITIALIZE SUBSCRIPT VECTOR
CALCULATE SUBSCRIPTS FOR NEXT PRODUCT
FORM PRODUCT & ADD TO ACCUMULATOR
90


90
LAST PRODUCT ?
NO
60
YES
SAVE NM
200
LAST COEFFICIENT ?
NO
500
YES
COMPLETE CALCULATIONS FOR NM, ALPHA
RETURN

NAME: CPT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear coefficients for bipolar junction transistor

VARIABLES:

A = Input value

AA = Intermediate calculations for ALPHA nonlinearity

ALPHA = Nonlinear coefficients

ALPHS = Intermediate calculations for ALPHA nonlinearity

AP = Multiplicative accumulator

C = Input value

CJ = Input value

CPRMF = Input value

FIJ = Subscripts for polynomial power series expansion coefficient

GAMMA = Nonlinear coefficients

HC2 = Nonlinear coefficients

PFEMX = Input value

PM = Nonlinear coefficients

PMO = Calculated parameter

IC = Input value

ICHMO = Calculated parameter

ICMAX = Input value

IE = Calculated parameter

K = Input value

KS = Nonlinear coefficients

MINDX = Incrementation vector for differentiation algorithm

MORD = Subscripts for differentiation algorithm

MU = Input value

N = Input value

QPRME = Intermediate calculation for HM nonlinearity

RATIO = Preliminary calculation

RP = Input value

RPI = Calculated parameter

RC = Input value

RCI = Calculated parameter

REF = Input value

REI = Calculated parameter

S = Additive accumulator

SCC = Input value

SCO = Calculated parameter

SIC = Preliminary calculation

SS = Additive accumulator

SWTCH = Control switch for 'FUN' algorithm

-1 = Initialize algorithm

0 = Continue algorithm

1 = Algorithm complete

VCB = Input value

VCBO = Input value

SUBROUTINES CALLED·

FUN

CALLING PROGRAMS·

PHASE1

DESCRIPTION·

Subroutine CPT calculates the device parameters for the bipolar junction transistor. These parameters include the coefficients for the four nonlinearities of the device model (KS, CAMMA, FM, and ALPHA) and the miscellaneous values HC2, ICHMØ, PE1, IF, PBI, PCI, C(2), and SCØ which are used in the admittance submatrix and current vector subroutines.

The parameters are derived from the input values stored in the first twenty words of the transistor data record which is transmitted to subroutine CPT in the common data buffer. Calculated device parameters are stored in the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

The code consists of two main sections. The miscellaneous device parameters, first order coefficients, and preliminary calculations which are derived from straightforward arithmetic expressions, are computed first. Then the higher order

coefficients are calculated recursively within a loop under the control of the index I which represents the order of the coefficients being calculated.

The calculation of coefficients for the KS nonlinearity uses the arithmetic statement function·

```
ALG10(X) = ALOG(X) * .4342944819
```

which computes log of a real argument. The following preliminary calculations are performed to simplify arithmetic expressions·

```
RATIO = ALG10(IC/ICMAX)
HMO = 1./(1.-(VCB/VCBO)**N)
IE = IC*(1.+HFEMX+A*RATIO*RATIO)/HMO/HFEMX
RE1 = RFF * .0256704l/IE
```

By definition the linear coefficient KS(1) is given by:

```
KS(1) = 1./RE1
```

Higher order coefficients are calculated recursively according to·

```
KS(I) = KS(I-1)/FLOAT(I) * RE1 * IE
```

The coefficients for the GAMMA nonlinearity are also calculated recursively, where·

```
GAMMA(1) = K/VCB**MU
```

and for higher order coefficients·

```
GAMMA(I) = -GAMMA(I-1) * (MU-FLOAT(I-2))/VCB/FLOAT(I)
```

The calculation of coefficients for the HM nonlinearity begins with·

```
HMO = 1./(1.-(VCB/VCBO)**N)
CPRME(1) = N * VCB**(N-1.)/VCBO**N
```

The linear coefficient HM(1) is a product of functions:

$$HM(1) = HMO * HMO * QPRME(1)$$

Successive higher order coefficients HM(I),I=2,3,...7 are calculated by differentiating the previous coefficient HM(I-1) with respect to VCB. The algorithm used to compute the higher order coefficients of the HM nonlinearity is derived from the rule of elementary calculus which defines the nth derivative of a product of functions to be a sum of products of lower order derivatives of those functions. In particular, the second order coefficient HM(2), obtained by differentiating HM(1), is a sum of products formulated as follows:

$$HM(2) = [HM(1)*HM(\emptyset)*QPRME(1)] + [HM(\emptyset)*HM(1)*QPRME(1)] + [HM(\emptyset)*HM(\emptyset)*QPRME(2)]$$

where HM(1) is the derivative of HM(Ø) and QPRME(2) is the derivative of QPRME(1). Note that each additive term is the product of two lower-order HM's and a QPRME. (The equivalency of HMO and HM(Ø) is forced by placing HMO and the HM vector in successive common storage locations.)

The third order coefficient HM(3), obtained by differentiating HM(2), consists of nine additive terms, where each term is the product of two lower-order HM's and a QPRME. In general each coefficient HM(I) consists of $3^{(I-1)}$ terms of the form HM * HM * QPRME where the highest derivative (subscript) of the HM's is (I-1).

In subroutine CPT these subscripts are stored in the integer vector MORD where MORD(1) is the subscript of the first HM in the product, MORD(2) is the subscript of the second HM, and MORD(3)

is the subscript of QPRME. In this context MORD= (Ø,Ø,1) represents the product HM(Ø)*HM(Ø)*QPRME(1). The 3 additive terms are generated under the control of the vector MINDX(L), L=1,2...(I-1) where each MINDX(L) ranges from 1 to 3. The elements of the MINDX and MORD vectors interact to formulate the complete expression for HM(I):

$$HM(I) = \sum_{MINDX(I-1)=1}^{3} \cdots \sum_{MINDX(2)=1}^{3} \sum_{MINDX(1)=1}^{3} HM(MORD(1)) * HM(MORD(2)) * QPRME(MORD(3))$$

The subscripts for each term are derived by incrementing the initial subscript vector MORD=(Ø,Ø,1) by MINDX(L) according to:

MORD(MINDX(L))=MORD(MINDX(L))+1 for L=1,2,...(I-1)

For example the calculation of HM(2) involves the single summation for MINDX(1) ranging from 1 to 3:

$$HM(2) = \sum_{MINDX(1)=1}^{3} HM(MORD(1)) * HM(MORD(2)) * QPRME(MORD(3))$$

The subscripts for each of the three additive terms are generated by incrementing MORD=(Ø,Ø,1) by MINDX as follows:

MINDX=(1) ➡ MORD(1,Ø,1) ➡ HM(1) * HM(Ø) * QPRME(1)

MINDX=(2) ➡ MORD(Ø,1,1) ➡ HM(Ø) * HM(1) * QPRME(1)

MINDX=(3) ➡ MORD(Ø,Ø,2) ➡ HM(Ø) * HM(Ø) * QPRME(2)

The calculation of HM(3) involves the double summation for

MINDX(1)=1 to 3 and MINDX(2)=1 to 3 and results in nine additive terms as follows:

MINDX=(1,1) ➡ MORD=(2,0,1) ➡ HM(2) * HM(0) * QPRME(1)
MINDX=(2,1) ➡ MORD=(1,1,1) ➡ HM(1) * HM(1) * QPRME(1)
MINDX=(3,1) ➡ MORD=(1,0,2) ➡ HM(1) * HM(0) * QPRME(2)
MINDX=(1,2) ➡ MORD=(1,1,1) ➡ HM(1) * HM(1) * QPRME(1)
MINDX=(2,2) ➡ MORD=(0,2,1) ➡ HM(0) * HM(2) * QPRME(1)
MINDX=(3,2) ➡ MORD=(0,1,2) ➡ HM(0) * HM(1) * QPRME(2)
MINDX=(1,3) ➡ MORD=(1,0,2) ➡ HM(1) * HM(0) * QPRME(2)
MINDX=(2,3) ➡ MORD=(0,1,2) ➡ HM(0) * HM(1) * QPRME(2)
MINDX=(3,3) ➡ MORD=(0,0,3) ➡ HM(0) * HM(0) * QPRME(3)

The algorithm used in subroutine CPT to compute HM(I) can be summarized as follows:

STEP 1: Initialization

a) Define II=I-1

b) Calculate QPRME(I)

c) Initialize additive accumulator S=0.

d) Initialize MINDX(L)=1 for L=1,2...II

STEP 2: Create One Additive Term

a) Initialize subscript vector MORD=(0,0,1)

b) Calculate subscripts by:

MORD(MINDX(L))=MORD(MINDX(L))+1

for L=1,2,...II

c) Form product:

HM(MORD(1)) * HM(MORD(2)) * QPRME(MORD(3))

and add to accumulator S

STEP 3: Update and Test MINDX

a) Initialize index T=0

b) Increment T=T+1. If T>II algorithm complete with HM(I) in S. Otherwise go to Step 3c.

c) Increment MINDX(T)=MINDX(T)+1. If MINDX(T)≤3, go to Step 2. Otherwise go to 3d.

d) Set MINDX(T)=1 and go to Step 3b.

The final computation for the HM nonlinearity is to divide each HM(I), I=1,2,...7 by I! which is accumulated in P:

HM(I) = HM(I)/P

The calculation of coefficients for the ALPHA nonlinearity begins with:

ALOGE = .4342944819

RATIO = ALG10(IC/ICMAX)

AA(1) = 1. + HFEMX + A * RATIO * (RATIO+2.*ALOGE)

By definition the first order coefficient ALPHA(1) is given by:

ALPHA(1) = 1./AA(1)

For higher order coefficients, each ALPHA(I), I=2,3,...7 is calculated according to the expression:

$$\text{ALPHA(I)} = -\frac{1}{\text{AA(1)}^{I}} \sum_{J=1}^{I-1} \text{ALPHA(J)}\ F_{I-1}^{J}(AA)$$

where $F_{I-1}^{J}(AA)$, a sum of products of AA's, is the polynomial power series expansion coefficient generated by subroutine FUN.

Each AA(I), I=2,3,...7 is calculated according to the expression:

AA(I) = SIC * (P * RATIO + Q * RATIO)

where:

$$\text{SIC} = \frac{2. * A * \text{LOGE}}{(I!)\ \ (IC)^{I-1}}$$

P = Q = 1. for I=2

Q = P - Q

P = -(I-1) * P } for I>2

Each F_{I-1}^{J} (AA) is computed by repeated calls to subroutine FUN under control of the variable SWTCH. For SWTCH=-1 the algorithm of subroutine FUN is initialized and each call returns a set of integers in the vector FIJ which determines the subscripts of the AA's to be multiplied together to form one

additive term of the expression $F^{J}_{I-1}(AA)$. Upon return from subroutine FUN, if SWTCH=Ø additional terms of the expression remain to be calculated and FUN is called again. When the expression is complete, SWTCH is returned as 1.

The algorithm used in subroutine CPT for calculating the sum

$$\sum_{J=1}^{I-1} \text{ALPHA}(J)\ F^{J}_{I-1}(AA)$$

can be summarized as follows.

STEP 1: Initialization

a) Define II=I-1

b) Initialize summation S=Ø.

c) Initialize index J=1

STEP 2: Calculate $F^{J}_{I-1}(AA)$

a) Initialize SWTCH=-1 and summation SS=Ø.

b) Call subroutine FUN to generate subscripts FIJ(L), L=1,2,...J for one additive term of the expression

c) Form the product.

$$AP = \prod_{L=1}^{J} AA\ (FIJ(L))$$

and add to summation SS.

d) If SWTCH=Ø, go to 2b. Otherwise expression is complete and stored in SS: go to Step 3.

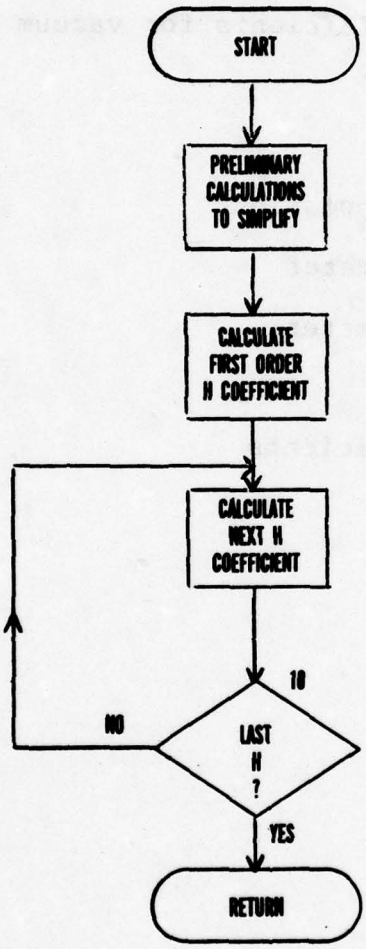
Step 3: Add New Term, Update and Test J

a) Form the product ALPHA(J)*SS and add to summation S.

b) Increment J=J+1. If J.LE.(I-1) go to Step 2. Otherwise algorithm complete with summation in S.

ALPHA(I) is then completed by multiplying the summation S by $-1/AA(1)^I$. The final computation for the ALPHA nonlinearity is to multiply each ALPHA(I), I=1,2,...7 by $HFBMX^I$ which is accumulated in Q:

ALPHA(I) = ALPHA(I) * Q


START
PRELIMINARY CALCULATIONS TO SIMPLIFY
CALCULATE FIRST ORDER H COEFFICIENT
CALCULATE NEXT H COEFFICIENT
LAST H ?
NO
YES
RETURN

NAME: CPVT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear coefficients for vacuum triode

VARIABLES:

A	= Preliminary calculation
G	= Calculated parameter
GP	= Calculated parameter
GO	= Input value
H	= Nonlinear coefficients
MU	= Input value
PHI	= Input value
VGO	= Input value
VGT	= Input value
VPO	= Input value

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

PHASE1

DESCRIPTION:

Subroutine CPVT calculates the device parameters for the vacuum triode. These parameters include the coefficients H for the vacuum triode's nonlinearity and the parameters G and GP which are used in the admittance submatrix and current vector subroutines.

The parameters are derived from the input values stored in the first nine words of the vacuum triode data record which is transmitted to subroutine CPVT in the common data buffer. The calculated device parameters are stored in the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

The parameters G and GP are given by:

```
G = GO * (1.-VCO/VGT)
GP = -GO/VGT
```

The preliminary calculation of A is performed to simplify the arithmetic:

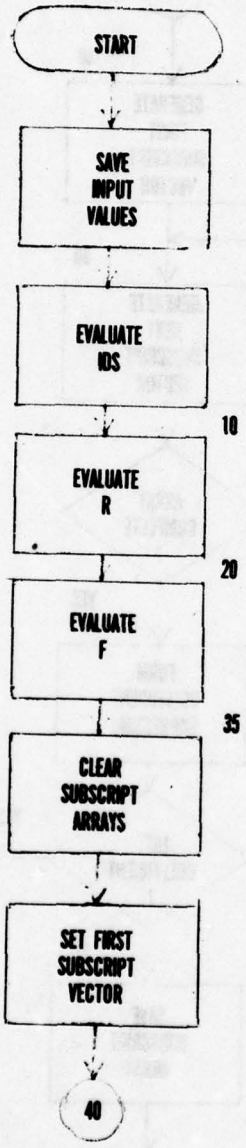
```
A = VGO + (VPO/MU) + PHI
```

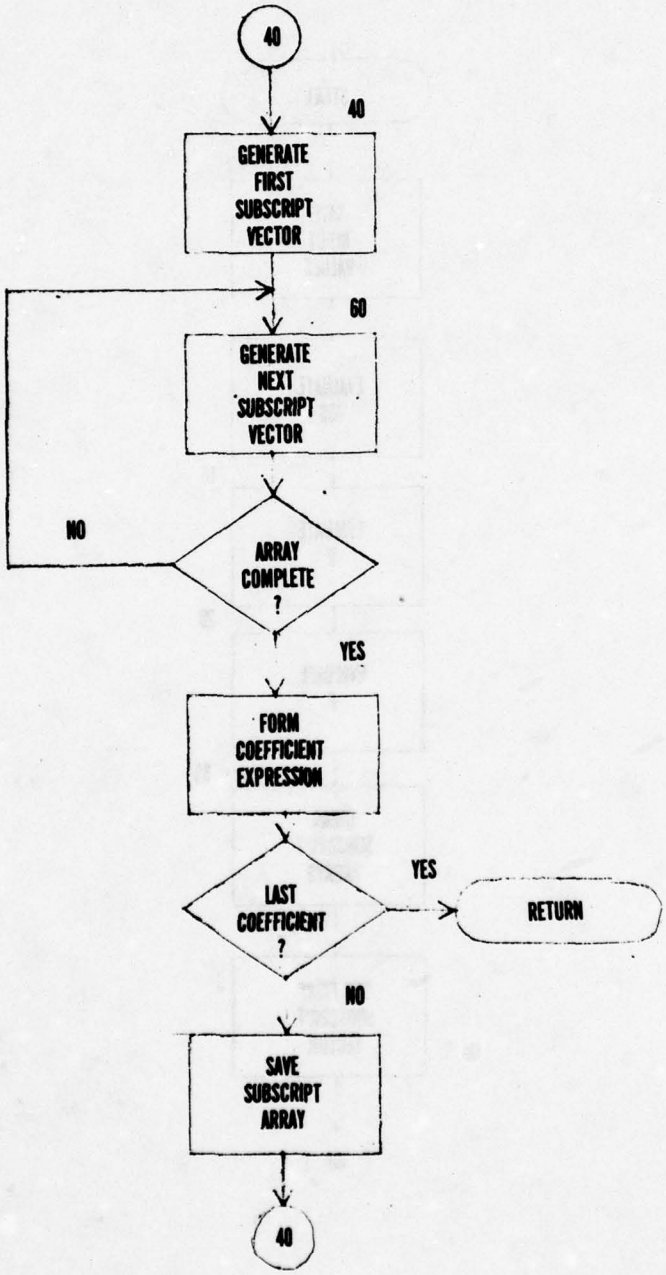
Then by definition the linear coefficient H(1) is given by:

```
H(1) = SQRT(A**3)
```

Higher order coefficients are calculated recursively according to:

$$H(I+1) = (2.5-I) * H(I)/(I*A)$$


START
SAVE INPUT VALUES
EVALUATE IDS
10
EVALUATE R
20
EVALUATE F
35
CLEAR SUBSCRIPT ARRAYS
SET FIRST SUBSCRIPT VECTOR
40


40
40
GENERATE FIRST SUBSCRIPT VECTOR
60
GENERATE NEXT SUBSCRIPT VECTOR
NO
ARRAY COMPLETE ?
YES
FORM COEFFICIENT EXPRESSION
LAST COEFFICIENT ?
YES
RETURN
NO
SAVE SUBSCRIPT ARRAY
40

NAME: DERIDS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Analytically generates nonlinear coefficients for junction field effect transistor

VARIABLES:

A	= Nonlinear coefficients
COEFR	= Intermediate calculation
DZ	= Intermediate calculation
EXPR	= Intermediate calculation
F	= d(IDS)/d(X)
FACT	= Accumulator for factorial
IDMAX	= Input value
IDS	= Value of function IDS
ISEQ	= Subscript array
KS	= Intermediate calculation
NO	= Order of derivative
PROD	= Multiplicative accumulator for derivative
Q	= Intermediate storage for input values
R	= d(X)/d(VGSI)
RHO	= Input value
RATIO	= Intermediate calculation
SEQ	= Subscript array
SUM	= Additive accumulator for derivative

VBI = Input value

VGSI = Input value

VP = Input value

X = Intermediate calculation

SUBROUTINES CALLED·

NONE

CALLING PROGRAMS·

CPMTJF

DESCRIPTION·

Subroutine DERIDS analytically generates the nonlinear coefficients A for the junction field effect transistor. Arguments transmitted to the subroutine are the JFET input values Q. The calculated coefficients are returned to the calling program in A.

The nonlinear coefficients A(N), N = 1,6 are defined as:

$$A(N) = \frac{1}{N!} \quad \frac{d^N(IDS)}{d(VGSI)}$$

and are calculated by repeatedly differentiating the function IDS with respect to VGSI, where:

$$IDS = 3*IDMAX*RHO \left[\frac{-RHO}{2} + \frac{1}{2}\left[RHO^2 + 4\left(\frac{1}{3} - RATIO + \frac{2}{3}(RATIO)^{3/2}\right)\right]^{1/2}\right]$$

and:

$$RATIO = \frac{VGSI + VBI}{VP + VBI}$$

First the function is simplified according to:

```
X = 1./3. - RATIO + 2./3. * (RATIO ** 1.5)
X = (X * 4.) + (RHO * RHO)
KS = 3./2. * IDMAX * RHO
IDS = (3. * IDMAX * RHO * RHO/2.) + (KS * X ** .5)
```

Then the first seven derivatives of the function X with respect to VGSI are calculated and stored in R. The first derivative is calculated directly according to:

```
DZ = 1./(VP + VBI)
COEFR = DZ * 4.
EXPR = .5
R(1) = COEFR * (-1. + RATIO ** EXPR)
```

Higher order derivatives R(I), I = 2,7 are calculated according to:

```
COEFR = COEFR * EXPR * DZ
EXPR = EXPR - 1.
R(I) = COEFR * RATIO ** EXPR
```

Next the first seven derivatives of IDS with respect to X are evaluated and stored in F. First:

```
N = .5
F(1) = KS * X ** N
```

The derivatives F(I), I = 2,8 are calculated recursively as:

```
F(I) = (N - (I-1) + 1.) * 1./X * F(I-1)
```

Finally the derivatives of IDS with respect to VGSI are evaluated and stored in A. The first derivative is given by:

```
A(1) = F(2) * R(1)
```

Each higher order derivative A(I), I = 2,6 is calculated by differentiating the previous A(I-1). The algorithm used to compute A(I) is derived from the rule of elementary calculus which defines the nth derivative of a product of functions to be a sum of products of derivatives of those functions. In particular, the second order derivative A(2) is a sum of products formulated as follows:

$$A(2) = [F(3) * R(1) * R(1)] + [F(2) * R(2)]$$

where F(3) * R(1) is the derivative of F(2) and R(2) is the derivative of R(1). Note that each additive term is the product of one F and one or more R's, and can therefore be differentiated according to the sum of products rule.

In subroutine DERIDS the subscripts of the F's and R's which constitute a derivative of IDS are developed in the integer array ISEQ, where each row of the ISEQ array contains the subscripts for one additive term of the derivative. The derivatives can then be represented by:

$$A(NO-1) = \sum_{J=1}^{K1} \left[F(ISEQ(1,J)) * \prod_{I=1}^{NO} R(ISEQ(I,J)) \right]$$

Note that the first row element of ISEQ maps to the subscript of F, while the remaining NO-1 elements map to the subscripts of R.

The elements of the ISEQ array are developed recursively. Once a derivative has been calculated, its ISEQ subscript array is saved in the array SEQ which serves as the basis for calculating the ISEQ subscript array for the next derivative.

The algorithm used by subroutine DERIDS for calculating the nonlinear coefficients A can be summarized as follows:

STEP 1: Initialization

a) Zero subscript arrays ISEQ and SEQ

b) Set first subscript vector: SEQ(1,1)=1

c) Initialize FACTorial product: FACT=1.

d) Initialize order of derivative: NO=2

e) Initialize SEQ row index: K=1

f) Initialize ISEQ row index: K1=1

STEP 2: Create First Additive Term

a) Initialize column index: MINDX=1

b) Calculate first two row elements:

ISEQ(1,K1) = SEQ(1,K) + 1

ISEQ(2,K1) = 1

c) If SEQ(L,K) = Ø for some L=2, NO, term complete.
Go to Step 3.

d) Otherwise, calculate next row element:

ISEQ(L+1,K1) = SEQ(L,K)

STEP 3: Calculate Next Additive Term

a) Increment column indes: MINDX=MINDX+1

b) Increment ISEQ row index: K1=K1 + 1

c) If SEQ(MINDX,K) = Ø, term complete.
Go to Step 4.

d) Otherwise calculate next row by:

ISEQ(L,K1) = SEQ(L,K) for L=1,NO

ISEQ(MINDX,K1) = ISEQ(MINDX,K1) + 1

e) Go to Step 3a.

STEP 4: Test for Subscript Array Complete

a) Increment SEQ row index: K=K+1

b) If SEQ(1,K) = Ø, array complete.
Go to Step 5.

c) Otherwise go to Step 2.

STEP 5: Form Coefficient Expression

a) Clear additive accumulator: SUM=Ø.

b) Initialize row index J=1

c) Initialize multiplicative accumulator:
PROD = F(ISEQ(1,J))

d) Form product:
PROD = PROD * R(ISEQ(I,J)) for I=2, NO

e) Accumulate sum of products:
SUM = SUM + PROD

f) Increment and test J. If J.GT. K1. e
expression complete. Go to Step 5g.
Otherwise go to Step 5c.

g) Calculate factorial: FACT=FACT * (NO-1)

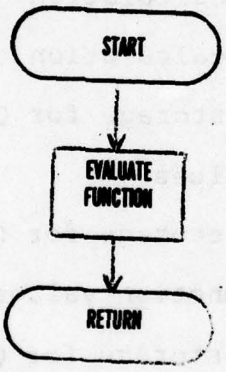
h) Store coefficient: A(NO-1) = SUM/FACT

STEP 6: Test for Algorithm Complete

a) Increment NO = NO + 1. If NO=7, algorithm complete.

b) Otherwise save subscript array in SEQ.

c) Go to Step 1e.


START
EVALUATE FUNCTION
RETURN

NAME: FIDS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Evaluates the function IDS for use in calculating coefficients for the junction field effect transistor

VARIABLES:

A	= Intermediate calculation
B	= Intermediate calculation
IDMAX	= Intermediate storage for Q(1)
Q	= JFET input values
RHO	= Intermediate storage for Q(2)
S	= Calculated function values
VBI	= Intermediate storage for Q(4)
VP	= Intermediate storage for Q(5)
X	= Operating point x-direction
Y	= Operating point y-direction

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

CPMTJF DERN

DESCRIPTION:

Subroutine FIDS evaluates the function IDS for use by the junction field effect transistor model. Arguments transmitted to the subroutine are: X and Y, the operating points, and Q, the transistor input values. Since IDS is a function of one variable, the second operating point is transmitted as Y=0. The calculated function IDS(X,Y) is returned to the calling program in S.

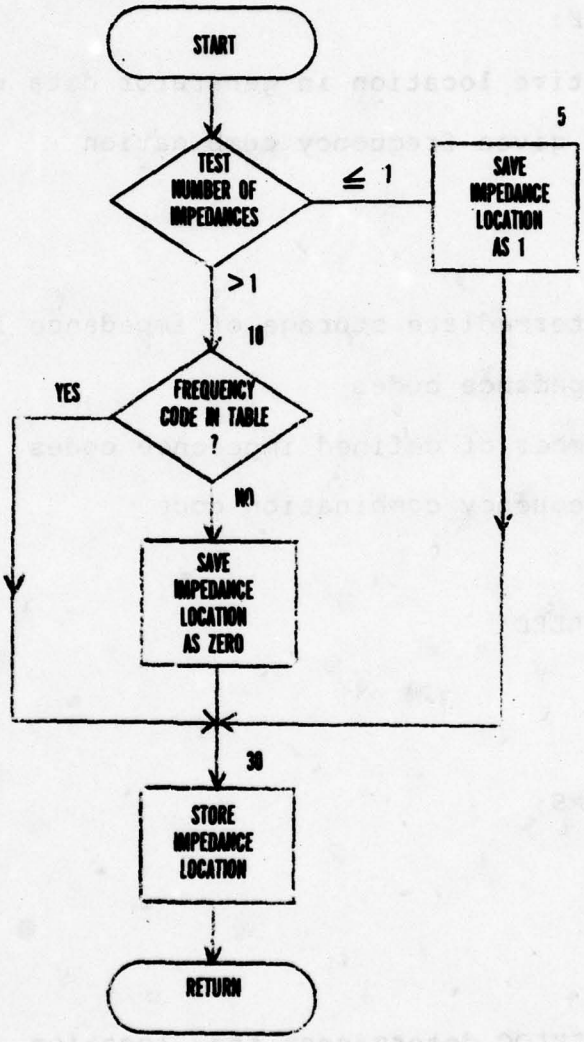
First the input values are stored as IDMAX=Q(1),RHO=Q(2), VP=Q(3), and VBI=Q(4). Next the expression for IDS is simplified by:

$$A = RHO$$

$$B = \frac{X + VBI}{VP + VBI}$$

Then IDS(X) is calculated according to:

$$IDS = 3 * A * IDMAX * (-\frac{A}{2}) * \frac{1}{2}\sqrt{4(\frac{1}{3} - B + \frac{2}{3}(B)^{\frac{3}{2}} + A^2}$$


START
TEST NUMBER OF IMPEDANCES
≤ 1
5
SAVE IMPEDANCE LOCATION AS 1
>1
10
YES
FREQUENCY CODE IN TABLE ?
NO
SAVE IMPEDANCE LOCATION AS ZERO
30
STORE IMPEDANCE LOCATION
RETURN

NAME: IMLOC

TYPE: FUNCTION

GENERAL PURPOSE:

Finds relative location in generator data record of impedance value for a given frequency combination

VARIABLES:

I = Intermediate storage of impedance location

IMTBL = Impedance codes

NIMPS = Number of defined impedance codes

SCODE = Frequency combination code

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

PHASE2

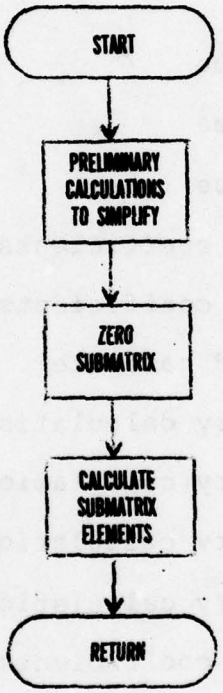
DESCRIPTION:

Function IMLOC determines the location of a generator's impedance (relative to the first impedance in the generator data record) for a particular frequency combination. Arguments transmitted to the function are: the frequency combination code, SCODE; a table of all frequency combination codes for which

impedances are stored in the generator data record, IMTBL; and the number of impedances in the data record, NIMPS. The impedance location is returned to the calling program in IMLOC.

Every NCAP generator has either a constant impedance over all frequencies or one impedance for each possible combination of its input frequencies. The impedances are converted to admittances in Phase Ø and stored with their corresponding frequency combination codes in the generator data record so that the Ith admittance in the record corresponds to the Ith frequency combination code in IMTBL.

For a generator with a constant impedance (NIMPS=1), the impedance location is simply IMLOC=1. Otherwise the location is determined by comparing SCODE to each frequency combination code IMTBL(I) for I=1, NIMPS. If a match is found, the table position I becomes the location returned in IMLOC. If no match is found, the location returned is IMLOC=Ø.


START
PRELIMINARY CALCULATIONS TO SIMPLIFY
ZERO SUBMATRIX
CALCULATE SUBMATRIX ELEMENTS
RETURN

NAME: MTPT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Create submatrix for vacuum pentode

VARIABLES:

CGC	= Input value
CPC	= Input value
CPG	= Input value
FP	= Nonlinear coefficients
F2	= Nonlinear coefficients
G	= Calculated parameter
GX1	= Preliminary calculation
GX2	= Preliminary calculation
GY1	= Preliminary calculation
GY2	= Preliminary calculation
P	= Nonlinear coefficients
MU	= Input value
P	= Submatrix

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

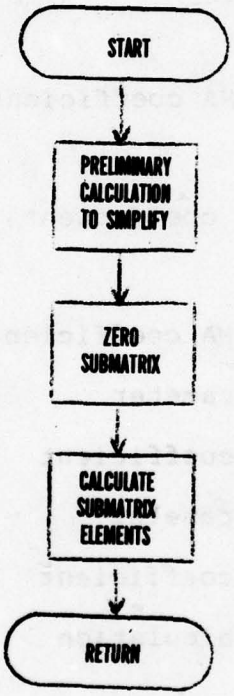
PHASE1

DESCRIPTION:

Subroutine MTPT calculates the complex admittances for the four-node vacuum pentode and stores them by columns in the submatrix PE(32). The submatrix elements are derived from the input data and device parameters stored in the pentode data record which is transmitted to subroutine MTPT through the common data buffer. The admittance submatrix is stored in the last 32 words of the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use for subsequent NCAP phases.

First the intermediate calculations for GX1, GX2, GY1, and GY2 are performed. The submatrix storage area is zeroed and the admittances are defined as follows:

REAL	IMAG	REAL	IMAG	REAL	IMAG	REAL	IMAG
P(1)=	P(2)=	P(9)=	P(10)=	P(17)=	P(18)=	P(25)=	P(26)=
0.	CGC+CPG	0.	0.	0.	-CPG	0.	-CGC
P(3)=	P(4)=	P(11)=	P(12)=	P(19)=	P(20)=	P(27)=	P(28)=
GX2	0.	GY2	0.	0.	0.	-GX2-GY2	0.
P(5)=	P(6)=	P(13)=	P(14)=	P(21)=	P(22)=	P(29)=	P(30)-
GX1	-CPG	GY1	0.	0.	CPG+CPC	-GX1-GY1	-CPC
P(7)=	P(8)=	P(15)=	P(16)=	P(23)=	P(24)=	P(31)=	P(32)=
-GX2-GX1	-CGC	-GY2-GY1	0.	0.	-CPC	GX2+GY2+ GX1+GY1	CGC+CPC


START
PRELIMINARY CALCULATION TO SIMPLIFY
ZERO SUBMATRIX
CALCULATE SUBMATRIX ELEMENTS
RETURN

NAME: MTT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Creates submatrix for bipolar junction transistor

VARIABLES:

ALPH1 = 1st order ALPHA coefficient

C1 = Input value

C2 = 1st order HC2 coefficient

C3 = Input value

CAM1 = 1st order GAMMA coefficient

HMO = Calculated parameter

HM1 = 1st order HM coefficient

ICHMO = Calculated parameter

K1 = 1st order KS coefficient

M1 = Preliminary calculation

PE = Submatrix

PEZ = Zero submatrix

PPI = Input value

PCI = Input value

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

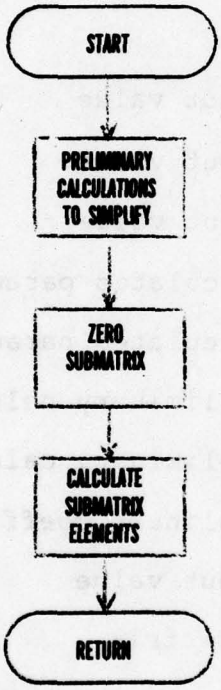
PHASE1

DESCRIPTION:

Subroutine MTT calculates the complex admittances for the four-node bipolar junction transistor and stores them in the complex submatrix PE(2,4,4). The submatrix elements are derived from the input values and device parameters stored in the transistor data record which is transmitted to subroutine MTT through the common data buffer. The admittance submatrix is stored in the last 32 words of the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

First the intermediate parameter M1 is calculated and the submatrix storage area is zeroed. The admittance submatrix elements are defined as follows:

Real	Imag	Real	Imag	Real	Imag	Real	Imag
PE(1,1,1) = RBI	PE(2,1,1) = C1+C3	PE(1,1,2) = -RBI	PE(2,1,2) = 0.	PE(1,1,3) = 0.	PE(2,1,3) = -C3	PE(1,1,4) = 0.	PE(2,1,4) = -C1
PE(1,2,1) = -RBI+M1	PE(2,2,1) = 0.	PE(1,2,2) = RBI+RCI+ K1*(1.- ALPH1*HMO)	PE(2,2,2) = C2+GAM1	PE(1,2,3) = -RCI-M1	PE(2,2,3) = -GAM1	PE(1,2,4) = K1*(ALPH1 *HMO-1.)	PE(2,2,4) -C2
PE(1,3,1) = -M1	PE(2,3,1) = -C3	PE(1,3,2) = -RCI+ ALPH1*HMO *K1	PE(2,3,2) = -GAM1	PE(1,3,3) = RCI+M1	PE(2,3,3) = C3+GAM1	PE(1,3,4) = -ALPH1* HMO*K1	PE(2,3,4) = 0.
PE(1,4,1) = 0.	PE(2,4,1) = -C1	PE(1,4,2) = -K1	PE(2,4,2) = -C2	PE(1,4,3) = 0.	PE(2,4,3) = 0.	PE(1,4,4) = K1	PE(2,4,4) = C1+C2


START
PRELIMINARY CALCULATIONS TO SIMPLIFY
ZERO SUBMATRIX
CALCULATE SUBMATRIX ELEMENTS
RETURN

NAME: MTVT

TYPE· SUBROUTINE

GENERAL PURPOSE·

Creates submatrix for vacuum triode

VARIABLES·

CGC	= Input value
CPC	= Input value
CPG	= Input value
G	= Calculated parameter
GP	= Calculated parameter
GX	= Preliminary calculation
GY	= Preliminary calculation
H	= Nonlinear coefficients
MU	= Input value
P	= Submatrix

SUBROUTINES CALLED·

NONE

CALLING PROGRAMS:

PHASE1

DESCRIPTION·

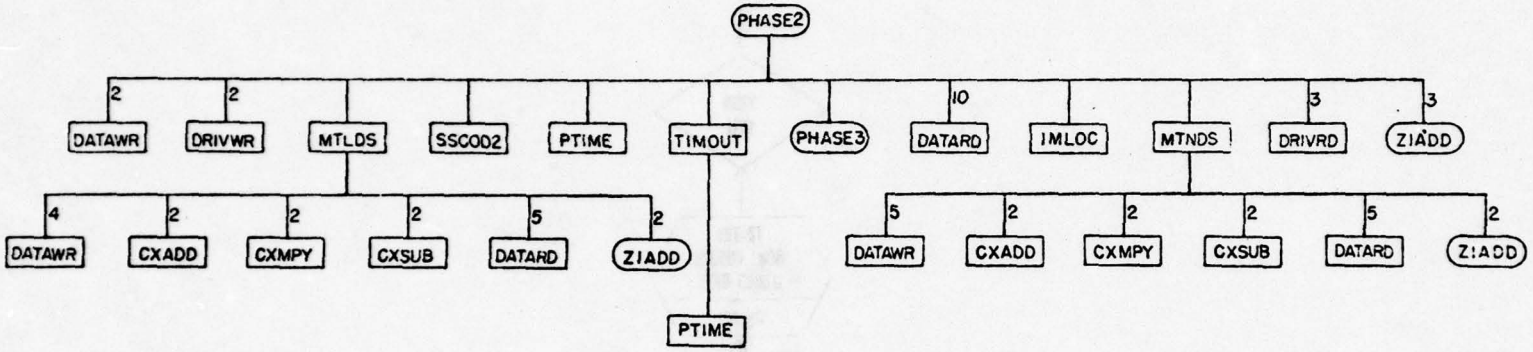
Subroutine MTVT calculates the complex admittances for the three-node vacuum triode and stores them by columns in the submatrix P(18). The submatrix elements are derived from the input values and device parameters stored in the vacuum triode data record which is transmitted to subroutine MTVT through the common data buffer. The admittance submatrix is stored in the last 18 words of the data record which is returned to subroutine PHASE1 and rewritten to file 20 for use by subsequent NCAP phases.

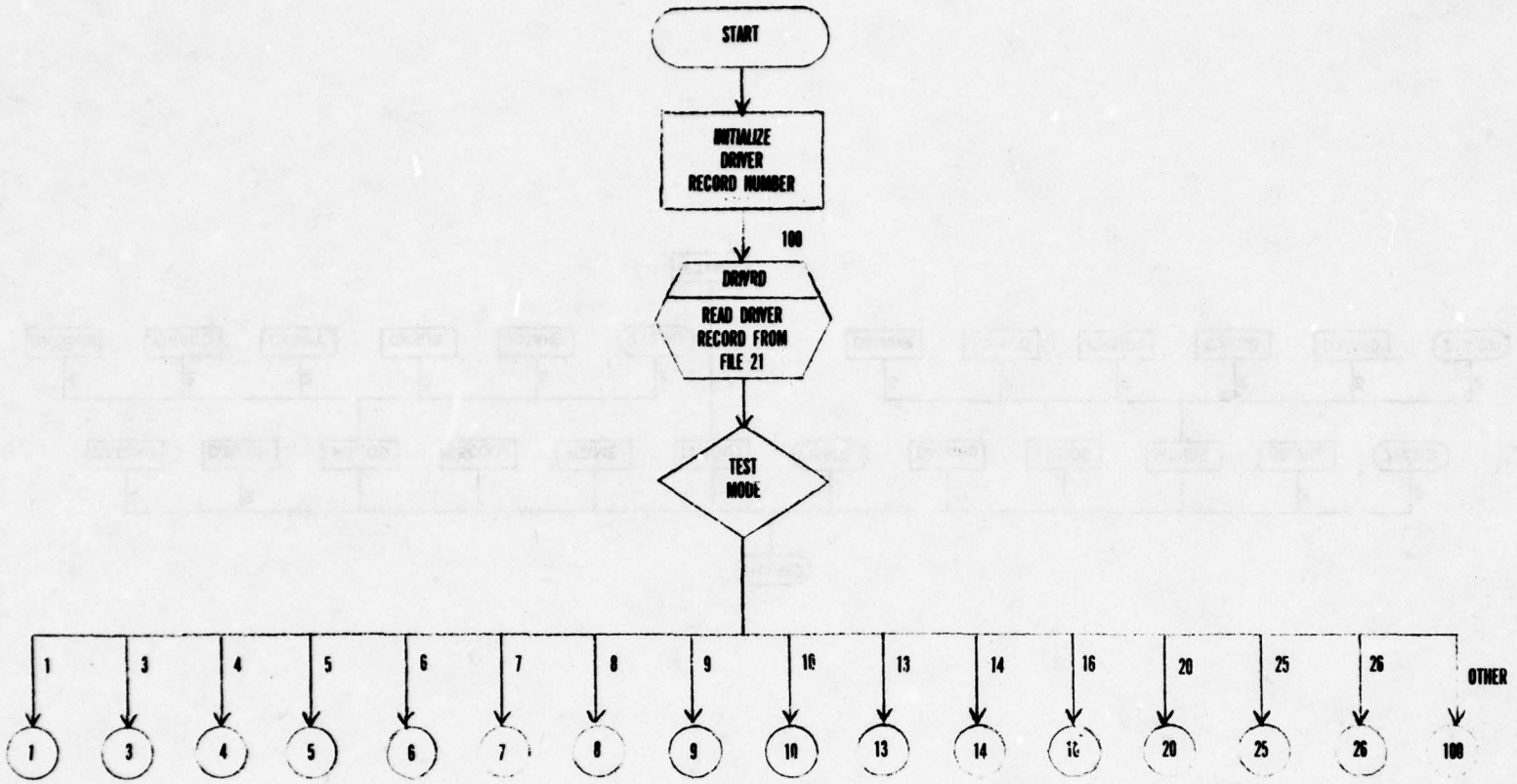
First the intermediate parameters GX and GY are calculated and the submatrix storage area is zeroed. The submatrix elements are defined as follows:

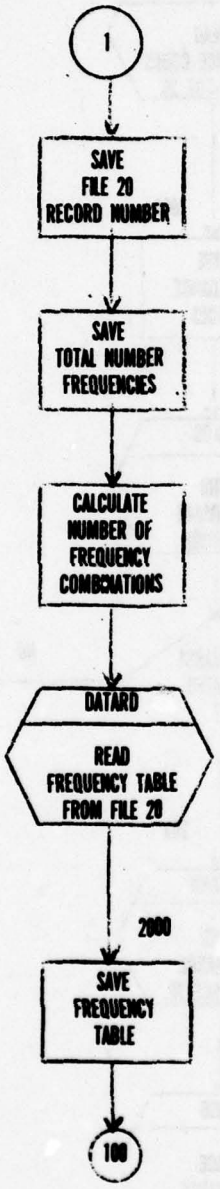
Real	Imag	Real	Imag	Real	Imag
P(1)= Ø	P(2)= CGC+CPG	P(7)= Ø	P(8)= -CPG	P(13)= Ø.	P(14)= -CGC
P(3)= GX	P(4)= -CPG	P(9)= GY	P(10)= CPG+CPC	P(15)= -GX-GY	P(16)= -CPC
P(5)= -GX	P(6)= -CGC	P(11)= GY	P(12)= -CPC	P(17)= -GX-GY	P(18)= CGC+CPC

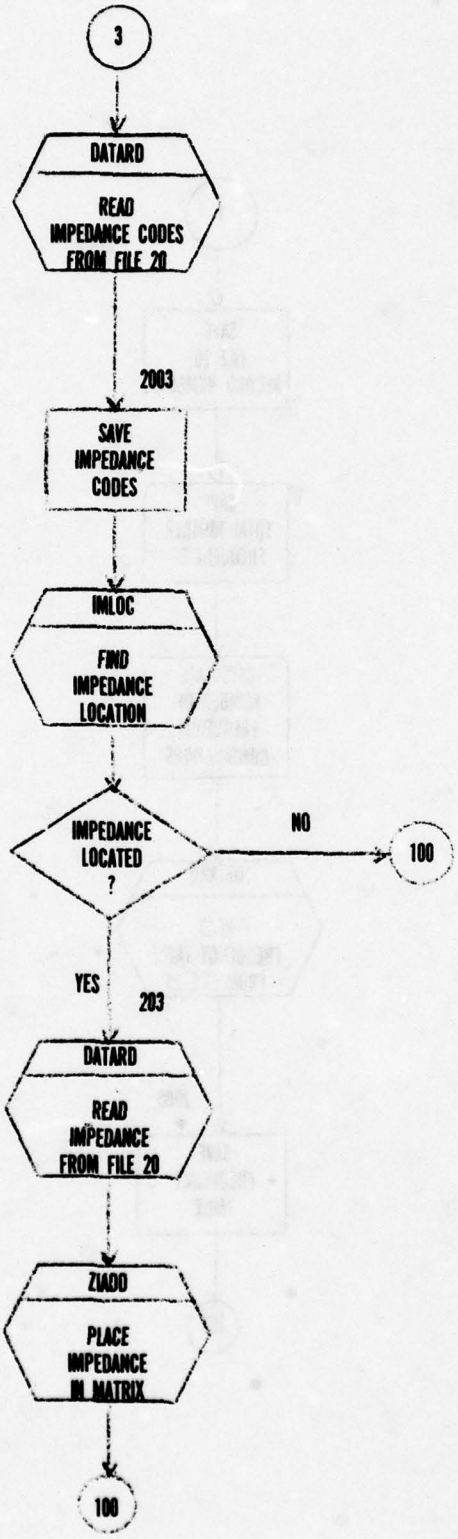
PHASE2

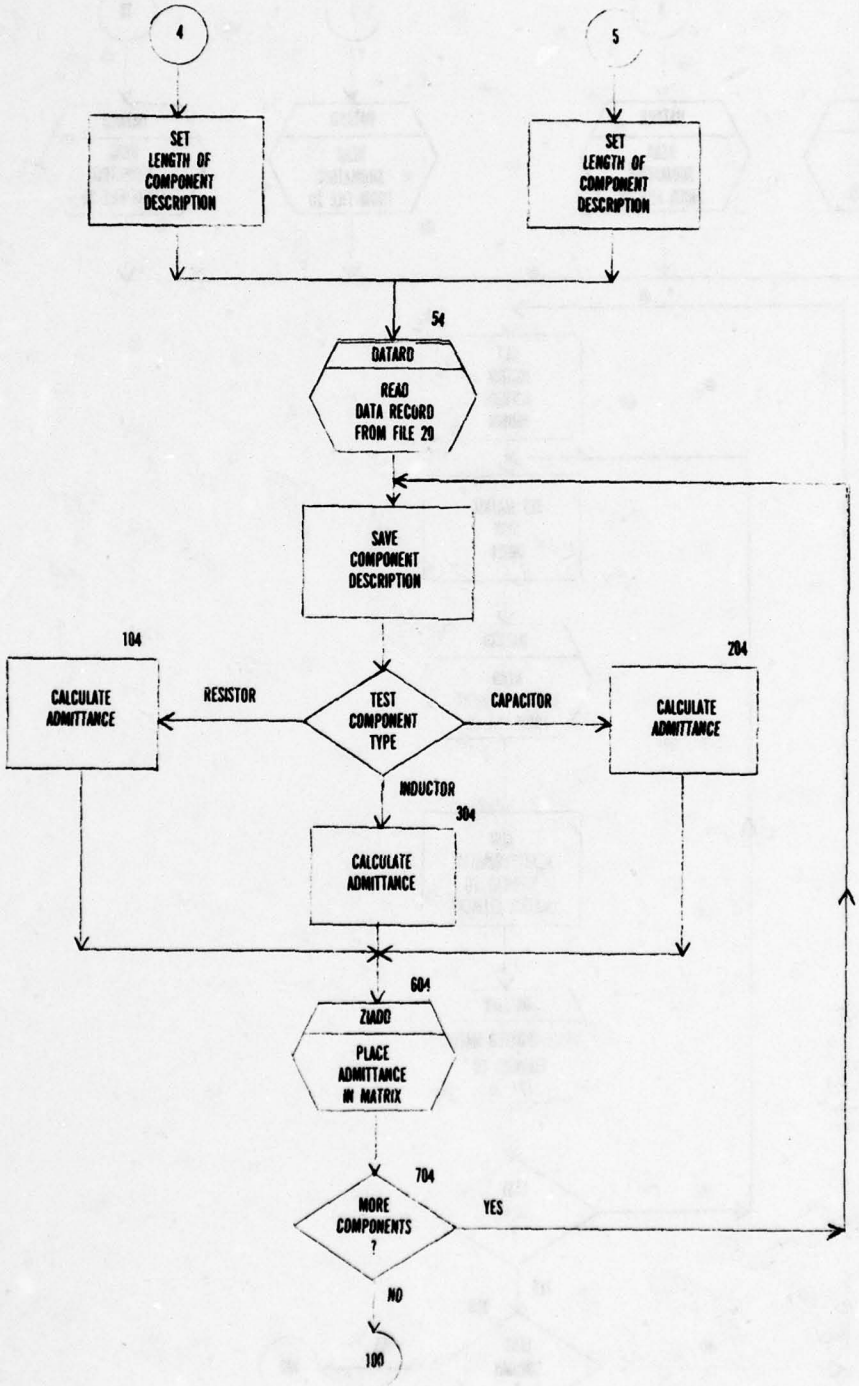
MTLDS
MTNDS
ZIADD

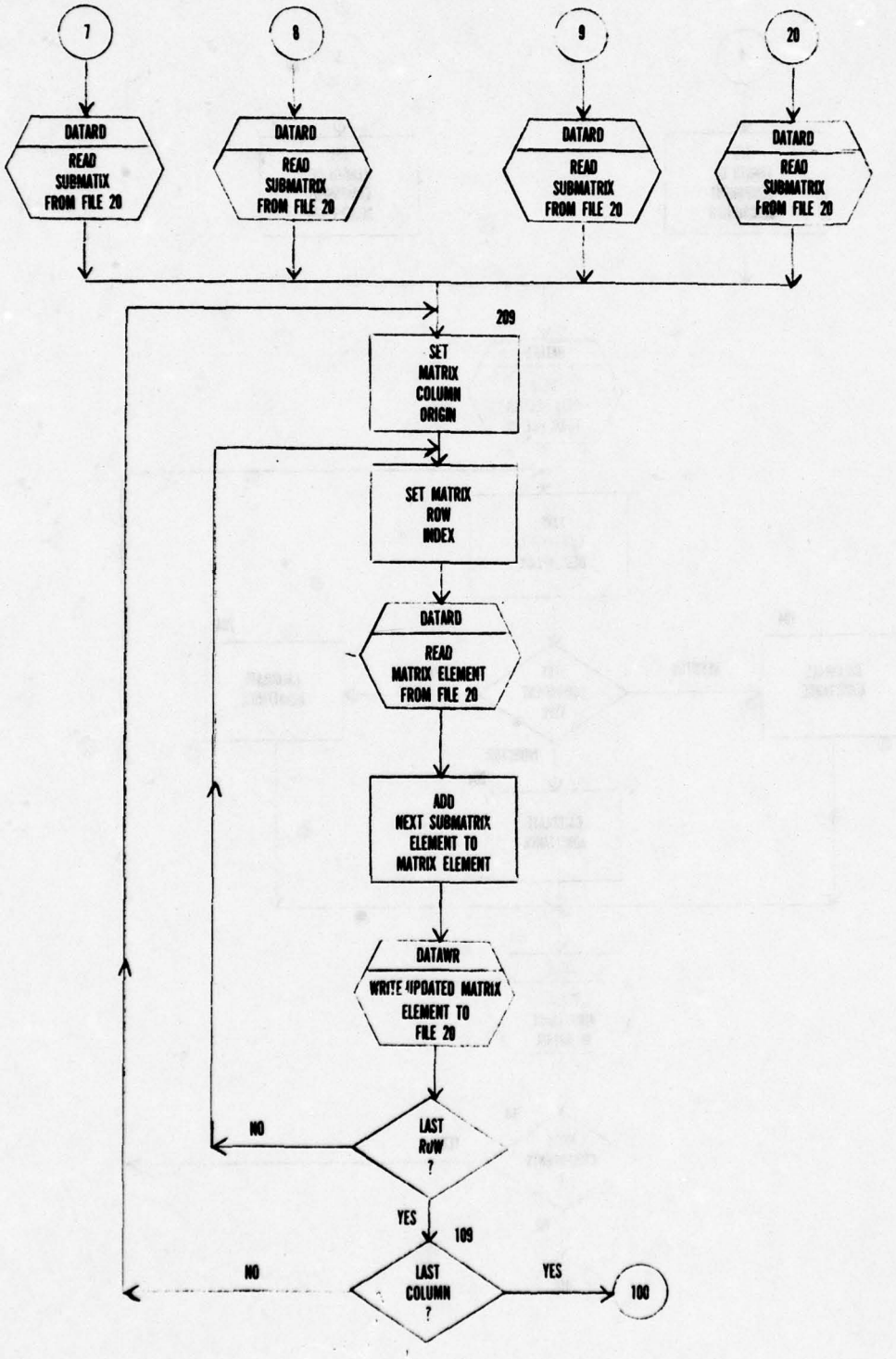

PHASE2
2
DATAWR
2
DRIVWR
MTLDS
SSCOD2
PTIME
TIMOUT
PHASE3
10
DATARD
IMLOC
MTNDS
3
DRIVRD
3
ZIADD
4
DATAWR
2
CXADD
2
CXMPY
2
CXSUB
5
DATARD
2
ZIADD
PTIME
5
DATAWR
2
CXADD
2
CXMPY
2
CXSUB
5
DATARD
2
ZIADD

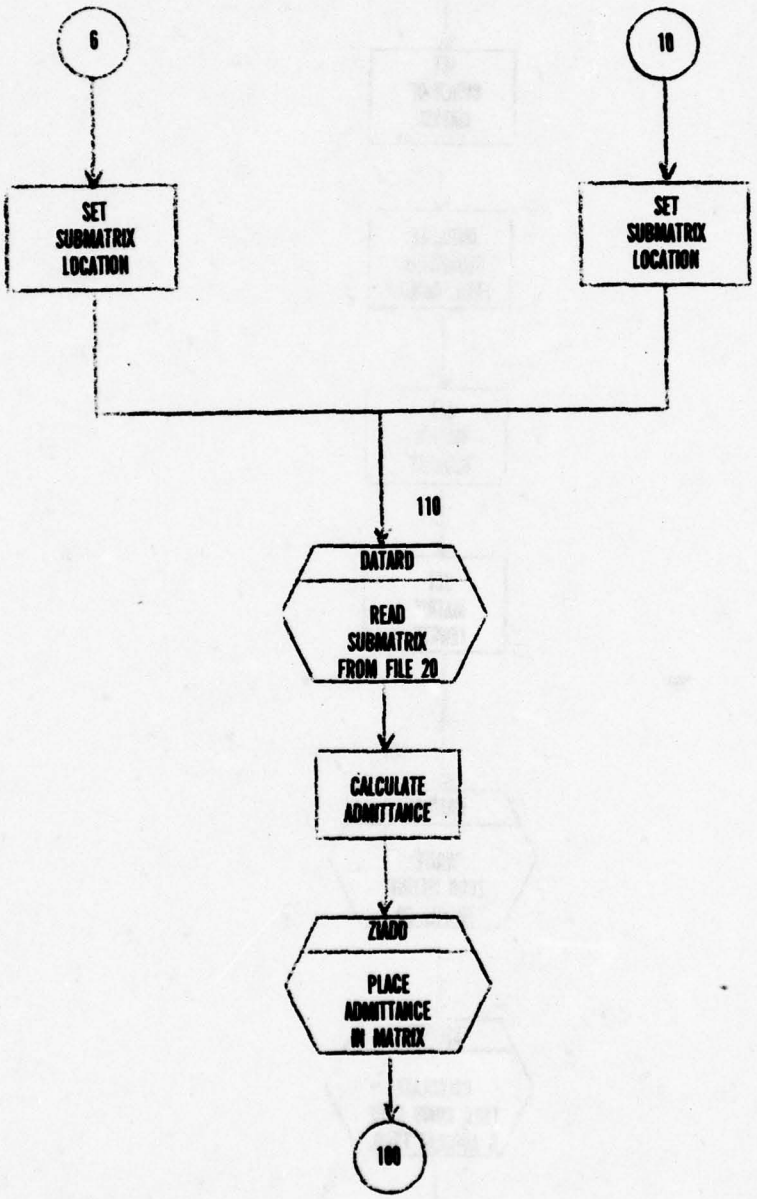

START
INITIALIZE DRIVER RECORD NUMBER
100
DRIVRD
READ DRIVER RECORD FROM FILE 21
TEST MODE
1
3
4
5
6
7
8
9
10
13
14
16
20
25
26
OTHER
1
3
4
5
6
7
8
9
10
13
14
16
20
25
26
100

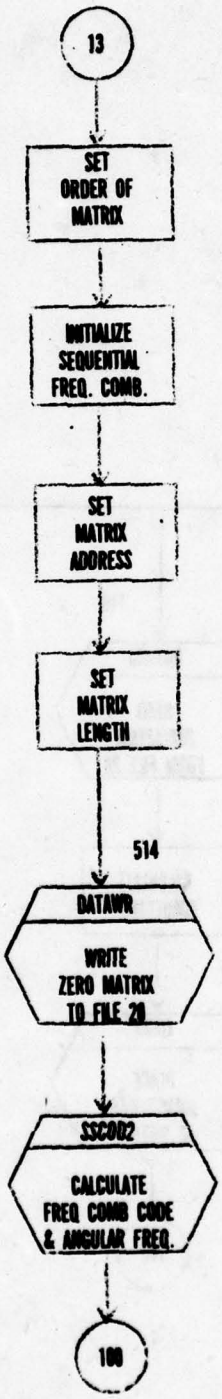

1
SAVE
FILE 20
RECORD NUMBER
SAVE
TOTAL NUMBER
FREQUENCIES
CALCULATE
NUMBER OF
FREQUENCY
COMBINATIONS
DATARD
READ
FREQUENCY TABLE
FROM FILE 20
2000
SAVE
FREQUENCY
TABLE
100

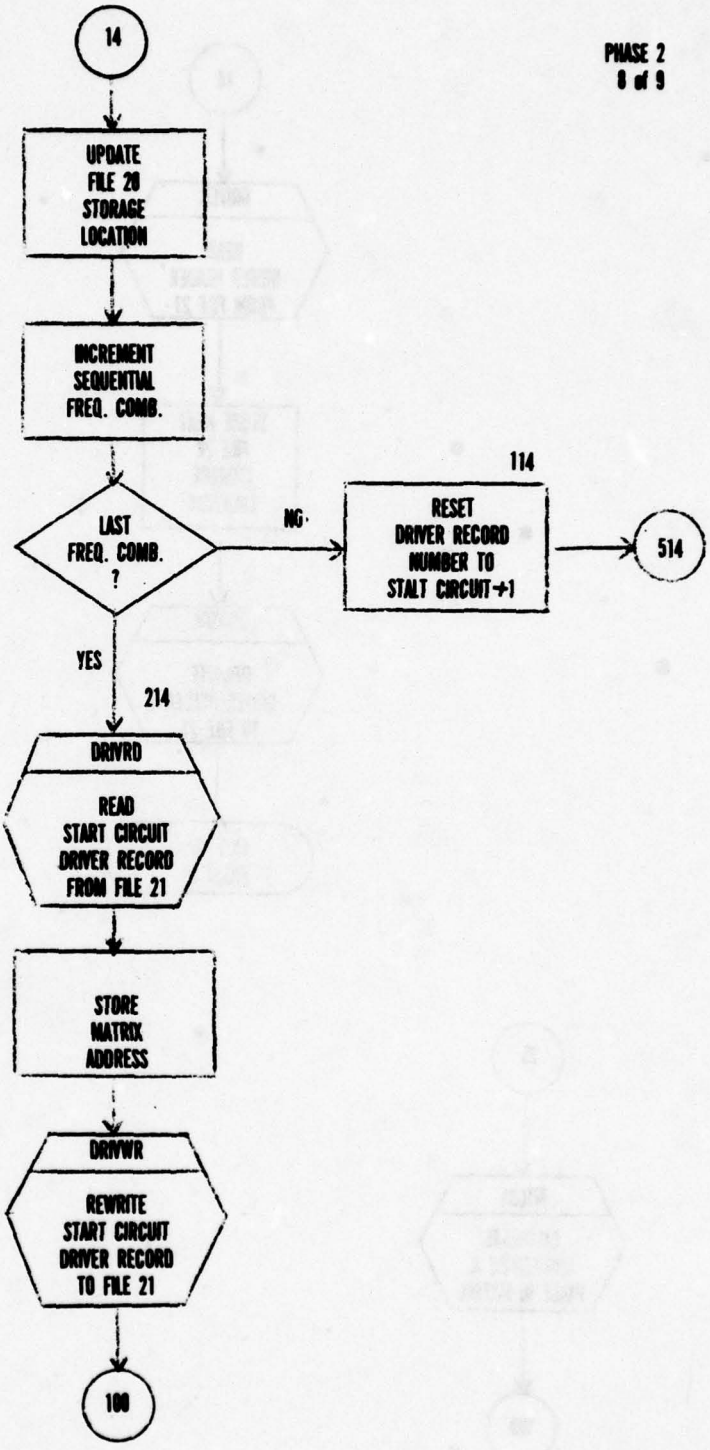

3
DATARD
READ IMPEDANCE CODES FROM FILE 20
2003
SAVE IMPEDANCE CODES
IMLOC
FIND IMPEDANCE LOCATION
IMPEDANCE LOCATED ?
NO
100
YES
203
DATARD
READ IMPEDANCE FROM FILE 20
ZIADD
PLACE IMPEDANCE IN MATRIX
100

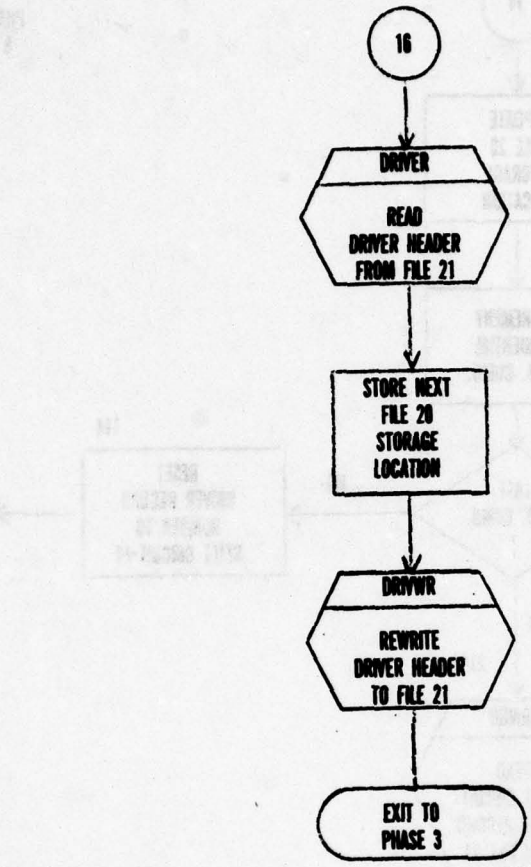

4
5
SET LENGTH OF COMPONENT DESCRIPTION
SET LENGTH OF COMPONENT DESCRIPTION
54
DATARD
READ DATA RECORD FROM FILE 29
SAVE COMPONENT DESCRIPTION
104
CALCULATE ADMITTANCE
RESISTOR
TEST COMPONENT TYPE
CAPACITOR
204
CALCULATE ADMITTANCE
INDUCTOR
304
CALCULATE ADMITTANCE
604
ZIADD
PLACE ADMITTANCE IN MATRIX
704
MORE COMPONENTS ?
YES
NO
100

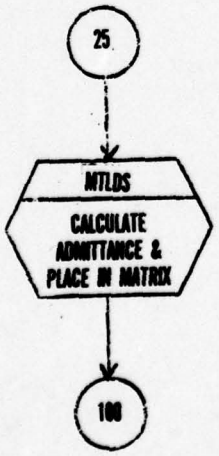

7
8
9
20
DATARD
READ SUBMATIX FROM FILE 20
DATARD
READ SUBMATRIX FROM FILE 20
DATARD
READ SUBMATRIX FROM FILE 20
DATARD
READ SUBMATRIX FROM FILE 20
209
SET MATRIX COLUMN ORIGIN
SET MATRIX ROW INDEX
DATARD
READ MATRIX ELEMENT FROM FILE 20
ADD NEXT SUBMATRIX ELEMENT TO MATRIX ELEMENT
DATAWR
WRITE UPDATED MATRIX ELEMENT TO FILE 20
LAST ROW ?
NO
YES
109
LAST COLUMN ?
NO
YES
100

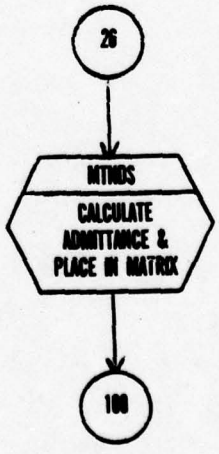

6
10
SET SUBMATRIX LOCATION
SET SUBMATRIX LOCATION
110
DATARD
READ SUBMATRIX FROM FILE 20
CALCULATE ADMITTANCE
ZIADD
PLACE ADMITTANCE IN MATRIX
100


13
SET
ORDER OF
MATRIX
INITIALIZE
SEQUENTIAL
FREQ. COMB.
SET
MATRIX
ADDRESS
SET
MATRIX
LENGTH
514
DATAWR
WRITE
ZERO MATRIX
TO FILE 20
SSC002
CALCULATE
FREQ COMB CODE
& ANGULAR FREQ.
100


14
UPDATE FILE 20 STORAGE LOCATION
INCREMENT SEQUENTIAL FREQ. COMB.
LAST FREQ. COMB. ?
NO
114
RESET DRIVER RECORD NUMBER TO STALT CIRCUIT+1
514
YES
214
DRIVRD
READ START CIRCUIT DRIVER RECORD FROM FILE 21
STORE MATRIX ADDRESS
DRIVWR
REWRITE START CIRCUIT DRIVER RECORD TO FILE 21
100


16
DRIVER
READ
DRIVER HEADER
FROM FILE 21
STORE NEXT
FILE 20
STORAGE
LOCATION
DRIVWR
REWRITE
DRIVER HEADER
TO FILE 21
EXIT TO
PHASE 3


25
MTLDS
CALCULATE
ADMITTANCE &
PLACE IN MATRIX
100


26
MTMDS
CALCULATE
ADMITTANCE &
PLACE IN MATRIX
100

NAME· PHASE2

TYPE· SUBROUTINE

GENERAL PURPOSE·

Construct admittance matrices

VARIABLES·

D = Complex admittance or admittance matrix element

FREQ = Frequency values

FRTBL = Frequency number table

IMTBG = Impedance codes

JRECS = Intermediate storage for driver file record number

LOCPE = Starting address of admittance matrix on file 20

LPE = Length of admittance matrix in words

NCOMP = Length of linear or nonlinear component data record

NF = Total number of defined frequencies

NNF = Total number of frequency combinations

S = Angular frequency

SCOMB = Sequential frequency combination

SCODE = Frequency combination code

SUBROUTINES CALLED:

DATARD DATAWR DRIVRD DRIVWR IMLOC

MTLDS MTNDS PHASE3 PTIME SSCOD2

TIMOUT ZIADD

CALLING PROGRAMS:

PHASE1

DESCRIPTION:

Subroutine PHASE 2 is the principal routine of NCAP's second computational phase. Its function is to construct and store the admittance matrices used in the circuit analysis. Because admittances are dependent upon frequency, one admittance matrix is constructed for each possible combination of the circuit's input frequencies.

An admittance matrix is a square array of complex numbers. The order (dimension) of an admittance matrix is determined by the number of nodes in the circuit. Each complex matrix element occupies two decimal storage locations, one for its real part and one for its imaginary part. The matrices are stored by columns on disk file 20. Every complex element of an admittance matrix is individually addressable. Successive admittance matrices are stored contiguously, in the order they are created, as one data record associated with the START CIRCUIT (MODE=13) driver record.

The construction of admittance matrices in subroutine PHASE2 is controlled by scanning the driver file $(2^{NF}-1)$ times, once for each possible combination of the NF input frequencies. The START

CIRCUIT driver record initiates the construction of the admittance matrix. During its construction in PHASE2 the matrix remains disk-resident. Whenever an admittance is to be placed in the matrix, the appropriate element is read from file 20, updated, and rewritten to its original location.

As the driver file scan proceeds, the admittances from circuit elements are placed in the matrix as their driver records are encountered. The END CIRCUIT driver record (MODE=14) terminates the matrix construction for a particular frequency combination and restarts the driver file scan for the next combination.

The first driver file scan begins by initializing the driver file record number to DRVREC=1. A driver record is read from file 21 by subroutine DRIVRD and subroutine PHASE2 responds to the MODE of the record as follows:

MODE=1 Driver File Header

The next available file 20 data storage location is set to DATREC=MISC(1) and the total number of input frequencies for the circuit is taken as NF=MISC(2). The number of possible frequency combinations is calculated according to

```
NNF=2**NF-1
```

The 20-word frequency table is read from file 20 by subroutine DATARD to the common data buffer BUFF. The frequency values are transferred from BUFF(1)-(10) to the decimal vector FREQ and the

frequency numbers are transferred from BUFF(11)-(20) to the integer vector FRTBL. Control then returns to the driver file scan.

MODE=13 Start Circuit

The order of the admittance matrix is defined by ORDER= MISC(4) and the sequential frequency combination is initialized to SCOMB=1. The disk address of the first admittance matrix is taken as LOCPE=DATREC and the length of each admittance matrix (in words) is calculated according to:

LPE=ORDER*ORDER*2

The admittance matrix storage area is cleared, one element at a time, by writing the complex number D=(0., 0.) to the LPE/2 successive file 20 locations beginning at DATREC. The frequency combination code, SCODE, and angular frequency, S, are calculated by subroutine SSCOD2 and control returns to the driver file scan.

MODE=3 Generator

The generator's impedance combination codes are read from file 20 by subroutine DATARD and transferred from the common data buffer BUFF to the integer vector IMTBG. The function IMLOC calculates the location of the admittance value corresponding to the frequency combination SCODE in the generator data record. If

function IMLOC returns an admittance location I.LE.0, there is no admittance for the present frequency combination, and control returns to the driver file scan without further processing.

Otherwise the disk address of the admittance is calculated and subroutine DATARD reads the complex admittance value from file 20 to the common data buffer BUFF. Subroutine ZIADD is called to place the generator's admittance in the matrix and control returns to the driver file scan.

MODE=4 Linear Components

MODE=5 Nonlinear Components

The number of words in each component description is defined as I4=4 for linear components or I4=13 for nonlinear components. The component data record is read from file 20 by subroutine DATARD to the common data buffer BUFF and the length of the data record is set by NCOMP=MISC(1).

For each component (I=1, NCOMP, I4) in the data record, the component type M is taken from the first word of its I4-word description, the nodes of connection K and L are taken from the second and third words, and the component value AA is taken from the fourth word. The component's admittance, a function of the input value and angular frequency S, is calculated and stored in D according to:

Resistor (M=1):

D(1)=1./(AA+1.E-7)

D(2)=0.

Capacitor (M=2):

D(1)=0.

D(2)=S*AA

Inductor (M=3):

D(1)=0.

D(2)=-1./(S*AA+1.E-7)

where the factor 1.E-7 prevents division by zero. ZIADD is then called to place the admittance in the matrix. After every component in the data record has been processed, control returns to the driver file scan.

MODE=6 Vacuum Diode

MODE=10 Semiconductor Diode

The relative location of the device admittance within the data record is set by N=13 for the vacuum diode or N=30 for the semiconductor diode. The admittance is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. After the imaginary part of the admittance, BUFF(2), is multiplied by the angular frequency S, subroutine ZIADD is called

to place the admittance in the matrix. Control then returns to the driver file scan.

MODE=7 Vacuum Triode

MODE=8 Vacuum Pentode

MODE=9 Bipolar Junction Transistor

MODE=20 Junction Field Effect Transistor

The device submatrix is read from file 20 by subroutine DATARD and stored by columns in the common data buffer BUFF. The submatrix is the nodal representation of the device's linear parts. For a device with N nodes, the submatrix is an NxN square array of complex elements. The admittance matrix is the nodal representation of the circuit's linear parts, where successive nodes are represented by rows and columns of like number. The nodes of the device model are numbered sequentially, beginning with the user-assigned base node MISC(3), and are represented in the admittance matrix by the NxN square subset of elements bounded by the MISC(3)rd row and column.

The device's admittances are placed in the matrix by adding the submatrix to the admittance matrix as follows. The first column of the submatrix is added to the MISC(3)rd column of the admittance matrix beginning at (MISC(3), MISC(3)). Then the second column of the submatrix is added to the (MISC(3)+1)st column of the admittance matrix beginning with (MISC(3), MISC(3)+1), and so on until the Nth column of the submatrix has been added to the (MISC(3)+N-1)st column of the admittance matrix.

The appropriate matrix elements are accessed by calculating their relative locations with respect to the origin of the admittance matrix. Matrix elements are complex numbers which occupy two storage locations each, and are stored and operated upon by columns. The locations of the NxN matrix elements bounded by the MISC(3)rd row and column are represented by PHASE2 variables as follows:

	MISC(3)	MISC(3)+1	...	MISC(3)+N-1
MISC(3)	I1	I1+I3	...	I2
MISC(3)+1	I1+2	I1+I3+2	...	I2+2
.	.	.		.
.	.	.		.
.	.	.		.
MISC(3)+N-1	I1+I4	I2+I3+I4	...	I2+I4

where:

I4=2*N-1

I3=2*ORDER

I1=(MISC(3)-1)*(ORDER+1)*2+1

I2=I1+ORDER*(I4-1)

N=Order of submatrix

ORDER=Order of admittance matrix

MISC(3)=Base node of connection

In the PHASE2 code, the index I addresses the column origins

and ranges from I1 to I2 in increments of I3. Within a column, the index J addresses successive row elements and ranges from I to I+I4 in increments of 2. The index K addresses the submatrix elements stored by columns in the data buffer BUFF. The algorithm for adding a device submatrix to the admittance matrix is summarizes as follows:

STEP 1: Initialization

a) Define I4, a constant depending on the number of nodes in the device model as follows:

MODE	DEVICE	NO. NODES	I4
7	Triode	3	5
8	Pentode	4	7
9	Transistor	4	7
20	JFET	4	7

b) Initialize submatrix index K= -1

c) Calculate I1, I2, and I3 for admittance matrix column origins

d) Initialize column origin I=I1

STEP 2: Operate on Ith Column

a) Set end of Ith column J2=I+I4.
Initialize row address J=I

b) Increment submatrix index K=K+2

c) Calculate absolute address of matrix element L=DATREC-1+J. Read element from file 20 to D and add submatrix element by:

D(1)=D(1)+BUFF(K)

D(2)=D(2)+BUFF(K+1)*S

Write updated matrix element to file 20.

d) Increment row address J=J+2. If J.LE.J2, go to Step 2b. Otherwise go to Step 3.

STEP 3: Increment & Test Column Origin

a) Increment column origin I=I+I3

b) If I.LE.I2, go to Step 2. Otherwise algorithm complete, return to driver file scan.

MODE=25 Linear Dependent Source

MODE=26 Nonlinear Dependent Source

The admittances for the linear and nonlinear dependent sources are calculated and placed in the admittance matrix by subroutines MTLDS and MTNDS respectively. Control then returns to the driver file scan.

MODE=14 End Circuit

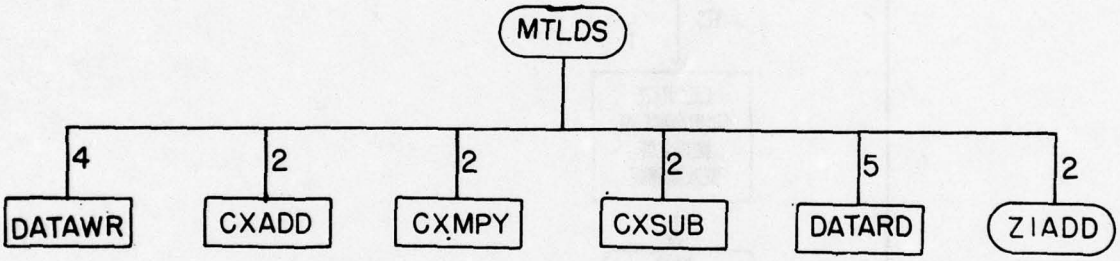
The next available file 20 storage location is calculated by DATREC=DATREC+LPE and the sequential frequency combination SCOMB is incremented. If the last admittance matrix has not been constructed (SCOMB.LE.NNF), the driver file record number is reset for the next driver file scan by DRVREC=MISC(1)+1. The admittance matrix storage area is cleared, the frequency combination code and angular frequency are calculated by subroutine SSCOD2, and control returns to the driver file scan.

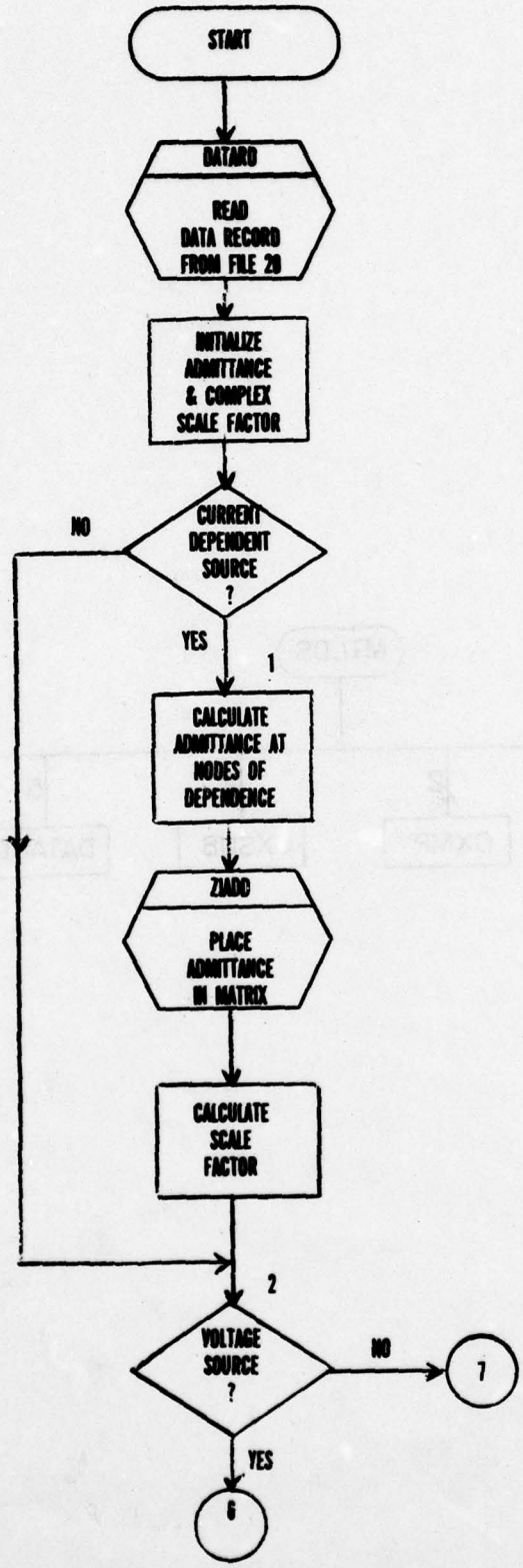
If the last admittance matrix has been constructed (SCOMB.GT.NNF), the matrix data record is attached to the start circuit driver record as follows: The present driver record number is saved in JRECS, and the driver record number is reset to the start circuit by DRVREC=MISC(1). The start circuit driver record is read from file 21 by subroutine DRIVRD, and after saving the matrix data location in MISC(5), the record is rewritten by subroutine DRIVWR. The driver record number is restored from JRECS and control returns to the driver file scan.

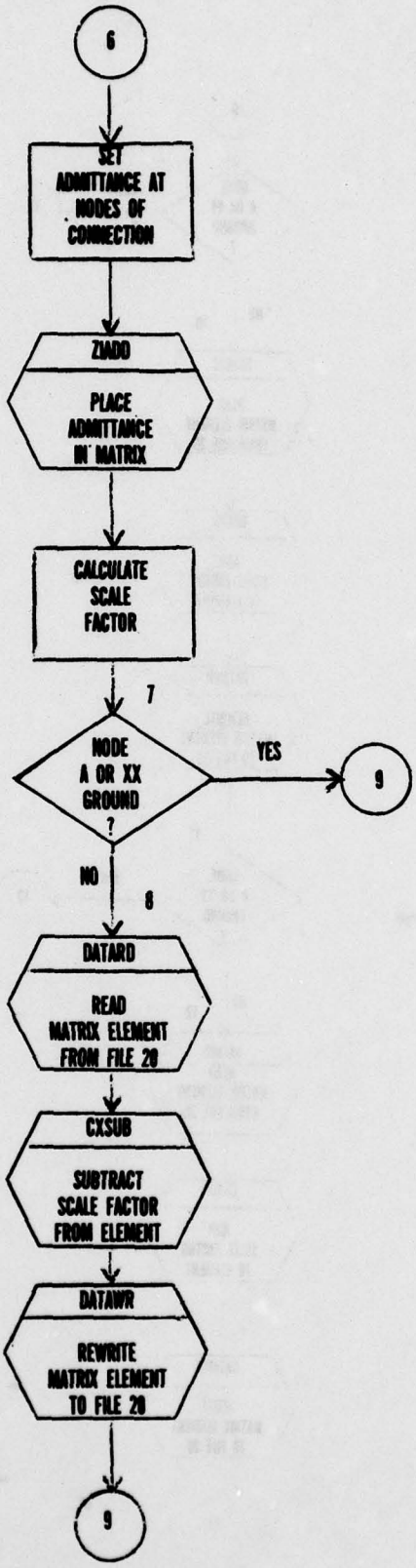
MODE=16 End

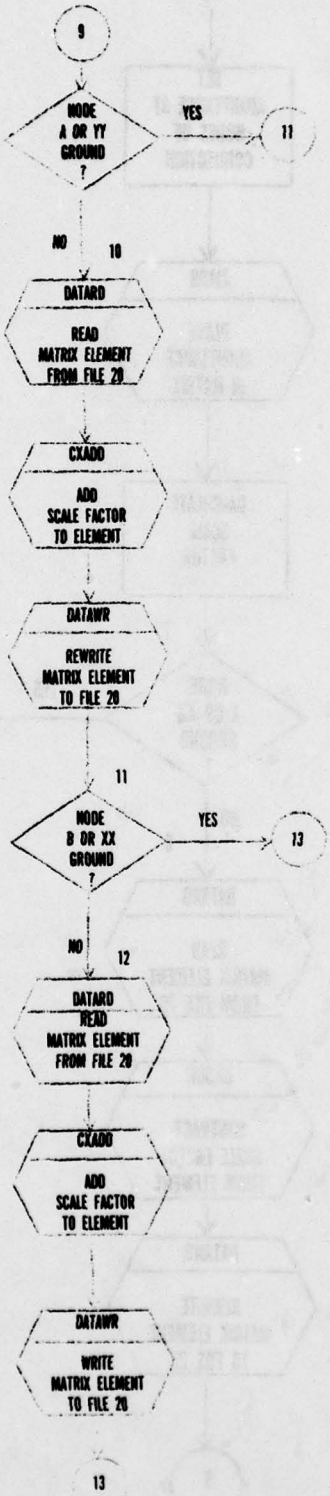
The driver file header is read by subroutine DRIVRD, updated to reflect the next available file 20 storage location, and rewritten by subroutine DRIVWR. Program control then transfers to subroutine PHASE3.

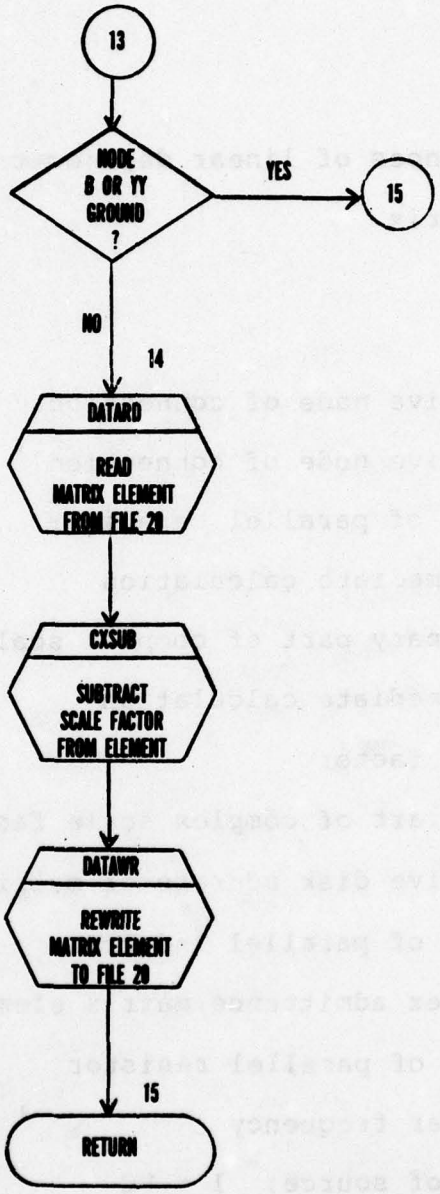
For other values of MODE, control returns directly to the driver file scan without further processing by subroutine PHASE2.


MTLDS
4
DATAWR
2
CXADD
2
CXMPY
2
CXSUB
5
DATARD
2
ZIADD


START
DATARD
READ DATA RECORD FROM FILE 20
INITIALIZE ADMITTANCE & COMPLEX SCALE FACTOR
CURRENT DEPENDENT SOURCE ?
NO
YES
1
CALCULATE ADMITTANCE AT NODES OF DEPENDENCE
ZIADD
PLACE ADMITTANCE IN MATRIX
CALCULATE SCALE FACTOR
2
VOLTAGE SOURCE ?
NO
7
YES
6


6
SET ADMITTANCE AT NODES OF CONNECTION
ZIADD
PLACE ADMITTANCE IN MATRIX
CALCULATE SCALE FACTOR
7
NODE A OR XX GROUND ?
YES
9
NO
8
DATARD
READ MATRIX ELEMENT FROM FILE 20
CXSUB
SUBTRACT SCALE FACTOR FROM ELEMENT
DATAWR
REWRITE MATRIX ELEMENT TO FILE 20
9


9
NODE A OR YY GROUND ?
YES
11
NO
10
DATARD
READ MATRIX ELEMENT FROM FILE 20
CXADD
ADD SCALE FACTOR TO ELEMENT
DATAWR
REWRITE MATRIX ELEMENT TO FILE 20
11
NODE B OR XX GROUND ?
YES
13
NO
12
DATARD
READ MATRIX ELEMENT FROM FILE 20
CXADD
ADD SCALE FACTOR TO ELEMENT
DATAWR
WRITE MATRIX ELEMENT TO FILE 20
13


13
MODE
8 OR YY
GROUND
?
YES
15
NO
14
DATARD
READ
MATRIX ELEMENT
FROM FILE 20
CXSUB
SUBTRACT
SCALE FACTOR
FROM ELEMENT
DATAWR
REWRITE
MATRIX ELEMENT
TO FILE 20
15
RETURN

NAME: MTLDS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Places admittances of linear dependent source in admittance matrix

VARIABLES:

A	= Positive node of connection
B	= Negative node of connection
C	= Value of parallel capacitor
CC	= Intermediate calculation
CI	= Imaginary part of complex scale factor
CL	= Intermediate calculation
CONST	= Scale factor
CR	= Real part of complex scale factor
I	= Relative disk address of matrix element
L	= Value of parallel inductor
PE	= Complex admittance matrix element
R	= Value of parallel resistor
S	= Angular frequency
SOURCE	= Type of source; 1 = VC
	2 = CC
	3 = VV
	4 = CV

X = Intermediate calculation

XX = Positive node of dependence

YY = Negative node of dependence

SUBROUTINES CALLED:

CXADD CXMPY CXSUB DATARD DATAWR

ZIADD

CALLING PROGRAMS:

PHASE2

DESCRIPTION:

Subroutine MTLDS calculates the admittances for the linear dependent source and places them in the admittance matrix. The linear dependent source data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The complex admittance is initialized to X=(1.,1.) and the scale factor is transferred from the data buffer to CONST.

Then for current dependent sources (SOURCE=2 or 4), the admittance at the nodes of dependence is derived from the values of the parallel components R, C, and L as follows·

X(1)=1./R for R≠0

X(2)=(S*C)-(1./S*L) for L≠0

where S is the angular frequency transmitted to subroutine MTLDS as an argument. The admittance at the nodes of dependence (XX and YY) is placed in the admittance matrix by subroutine ZIADD. Subroutine CXMPY multiplies the scale factor by the admittance

and the result is stored in CONST.

For current sources (SOURCE=3 or 4), the admittance at the nodes of connection (A and B) is defined as X=(1.ØE1Ø., Ø.) and placed in the admittance matrix by subroutine ZIADD. Subroutine CXMPY multiplies the scale factor by the admittance and the result is stored in CONST.

Then the calculated scale factor CONST is placed in the admittance matrix by subtracting it from the matrix elements (A,XX) and (B,YY) and adding it the elements (A,YY) and (B,XX). If neither node A nor XX is ground (A*XX≠Ø), the location of the element (A,XX) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

I= LOCATE(A,XX)= 2*((XX-1)*ORDER+A)-1

where ORDER is the number of elements in each column of the matrix transmitted to subroutine MTLDS through global common. The absolute address is computed by I=DATREC-1+I and the element is read from file 2Ø by subroutine DATARD and stored in PE. The complex scale factor is subtracted from PE by subroutine CXSUB and the updated matrix element is rewritten to file 2Ø by subroutine DATAWR.

If neither node A nor YY is ground (A*YY≠Ø), the location of the element (A,YY) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

I= LOCATE(A,YY)= 2*((YY-1)*ORDER+A)-1

The absolute address is computed by I=DATREC-1+I and the element is read from file 2Ø by subroutine DATARD and stored in PE. The complex scale factor is added to PE by subroutine CXADD and the

updated matrix element is rewritten to file 20 by subroutine DATAWR.

If neither node B nor XX is ground (B*XX≠0), the location of the element (B,XX) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

I= LOCATE(B,XX)= 2*((XX-1)*ORDER+B)-1

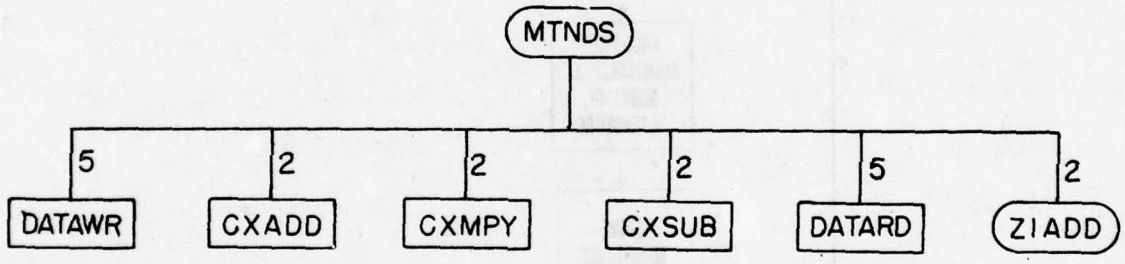
The absolute address is computed by I=DATREC-1+I and the element is read from file 20 by subroutine DATARD and stored in PE. The complex scale factor is added to PE by subroutine CXADD and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

If neither node B nor YY is ground (B*YY≠0), the location of the element (B,YY) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

I= LOCATE(B,YY)= 2*((YY-1)*ORDER+B)-1

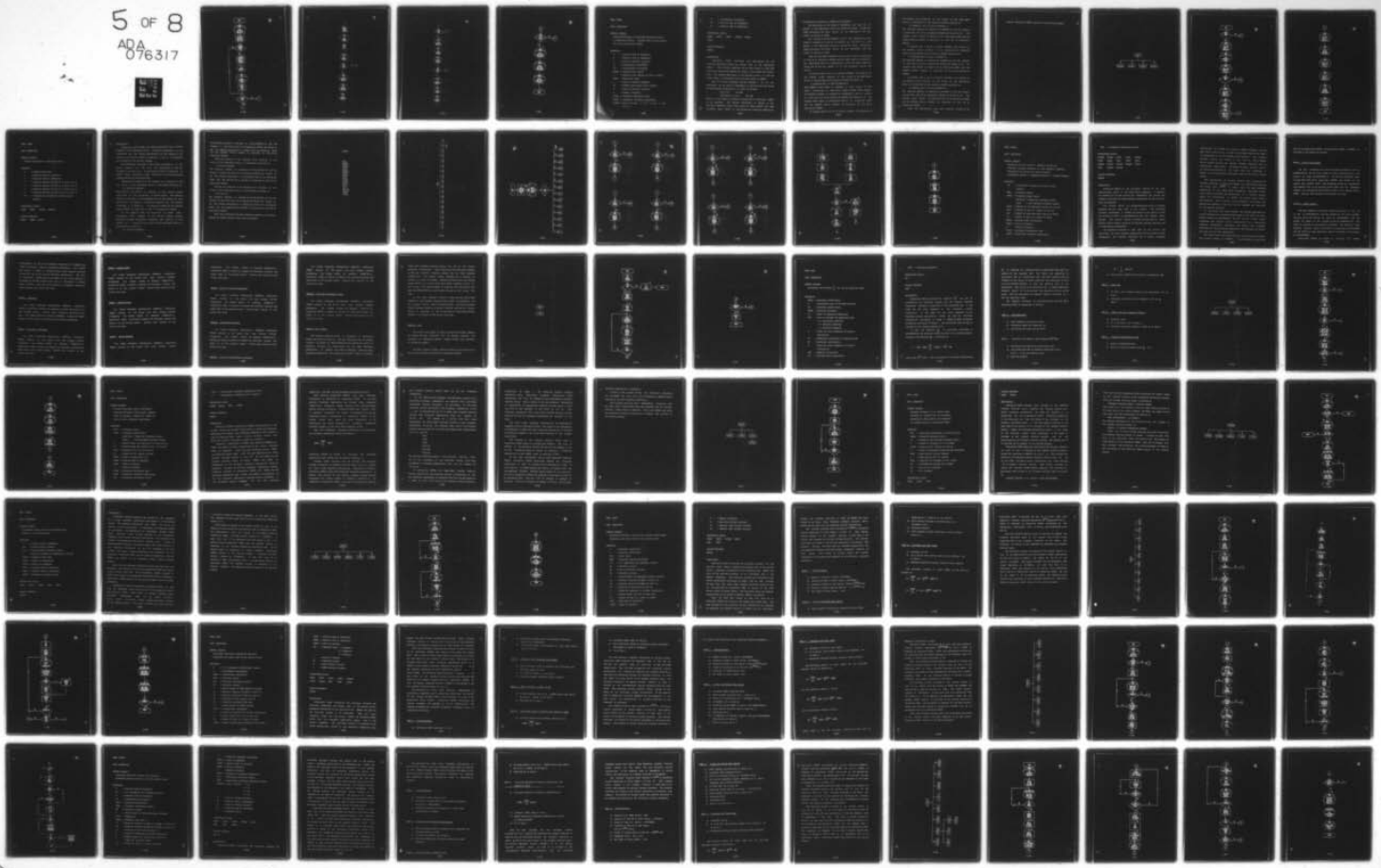
The absolute address is computed by I=DATREC-1+I and the element is read from file 20 by subroutine DATARD and stored in PE. The complex scale factor is subtracted from PE by subroutine CXSUB and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

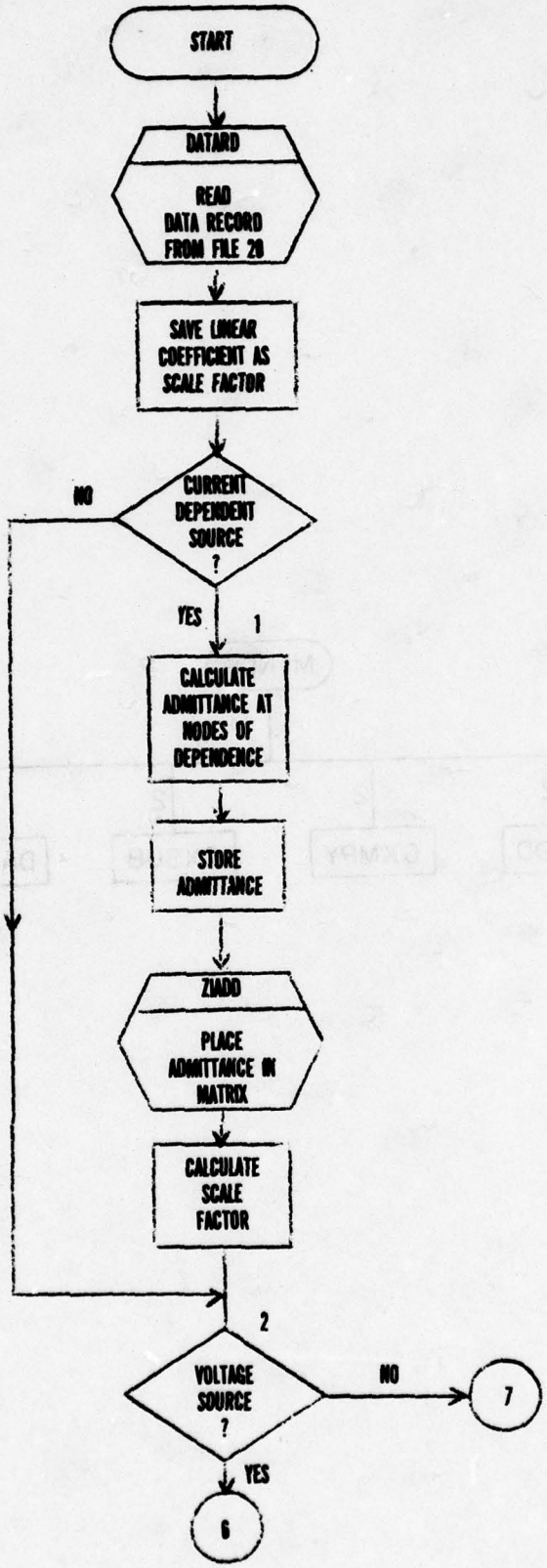
After the admittances have been properly placed in the matrix, subroutine MTLDS returns to the calling program.

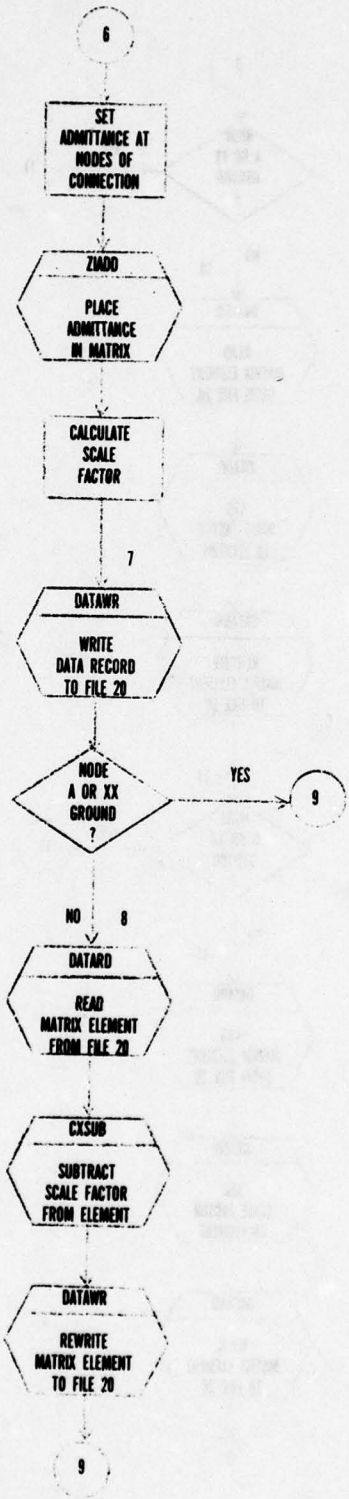

MTNDS
5
DATAWR
2
CXADD
2
CXMPY
2
CXSUB
5
DATARD
2
ZIADD

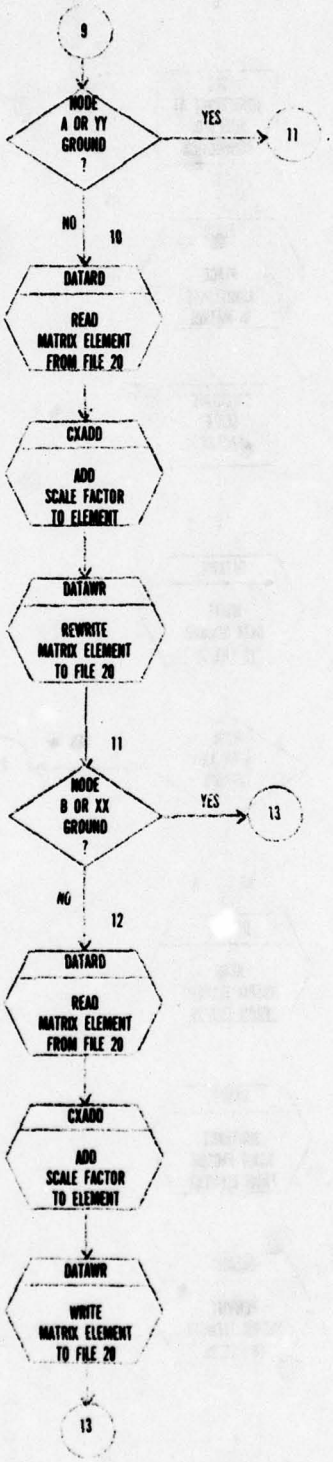
AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

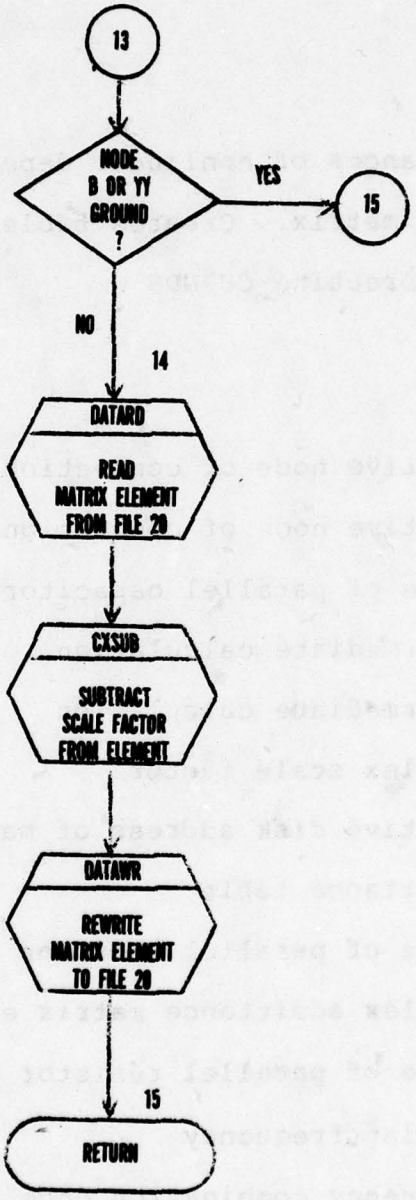
5 OF 8
ADA 076317




START
DATARD
READ DATA RECORD FROM FILE 28
SAVE LINEAR COEFFICIENT AS SCALE FACTOR
CURRENT DEPENDENT SOURCE ?
NO
YES
1
CALCULATE ADMITTANCE AT NODES OF DEPENDENCE
STORE ADMITTANCE
ZIADD
PLACE ADMITTANCE IN MATRIX
CALCULATE SCALE FACTOR
2
VOLTAGE SOURCE ?
NO
7
YES
6


6
SET ADMITTANCE AT NODES OF CONNECTION
ZIADD
PLACE ADMITTANCE IN MATRIX
CALCULATE SCALE FACTOR
7
DATAWR
WRITE DATA RECORD TO FILE 20
NODE A OR XX GROUND ?
YES
9
NO
8
DATARD
READ MATRIX ELEMENT FROM FILE 20
CXSUB
SUBTRACT SCALE FACTOR FROM ELEMENT
DATAWR
REWRITE MATRIX ELEMENT TO FILE 20
9


9
NODE A OR YY GROUND ?
YES
11
NO
10
DATARD
READ MATRIX ELEMENT FROM FILE 20
CXADD
ADD SCALE FACTOR TO ELEMENT
DATAWR
REWRITE MATRIX ELEMENT TO FILE 20
11
NODE B OR XX GROUND ?
YES
13
NO
12
DATARD
READ MATRIX ELEMENT FROM FILE 20
CXADD
ADD SCALE FACTOR TO ELEMENT
DATAWR
WRITE MATRIX ELEMENT TO FILE 20
13


13
MODE B OR YY GROUND ?
YES
15
NO
14
DATARD
READ MATRIX ELEMENT FROM FILE 20
CXSUB
SUBTRACT SCALE FACTOR FROM ELEMENT
DATAWR
REWRITE MATRIX ELEMENT TO FILE 20
15
RETURN

NAME: MTNDS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Places admittances of nonlinear dependent source in admittance matrix. Creates table of admittances for use by Subroutine CURNDS

VARIABLES:

A = Positive node of connection

B = Negative node of connection

C = Value of parallel capacitor

CC = Intermediate calculation

CL = Intermediate calculation

CONST = Complex scale factor

I = Relative disk address of matrix element

IMPS = Admittance table

L = Value of parallel inductor

PE = Complex admittance matrix element

R = Value of parallel resistor

S = Angular frequency

SCODE = Frequency combination code

SCOMB = Sequential frequency combination

SOURCE = Type of source: 1 = VC, 2 = CC, 3 = VV, 4 = CV

X = Intermediate calculation

XX = Positive node of dependence

YY = Negative node of dependence

SUBROUTINES CALLED:

CXADD CXMPY CXSUB DATARD DATAWR

ZIADD

CALLING PROGRAMS:

PHASE2

DESCRIPTION:

Subroutine MTNDS calculates the admittances for the nonlinear dependent source and places them in the admittance matrix. The nonlinear dependent source data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The complex admittance is initialized to X=(1.,1.) and the scale factor is transferred from the data buffer to CONST.

Then for current dependent sources (SOURCE= 2 or 4), the admittance at the nodes of dependence is derived from the values of the parallel components R, C, and L as follows:

X(1)=1./R for R≠0

X(2)=(S*C)-(1./S*L) for L≠0

where S is the angular frequency transmitted to subroutine MTNDS as an argument. The complex admittance is stored in the nonlinear dependent source data record at IMPS(1,SCOMB) and IMPS (2,SCOMB) where SCOMB is the sequential frequency combination

transmitted to subroutine MTNDS as an argument.

The admittance at the nodes of dependence (XX and YY) is placed in the admittance matrix by subroutine ZIADD. Subroutine CXMPY multiplies the scale factor by the admittance and the result is stored in CONST.

For current sources (SOURCE= 3 or 4), the admittance at the nodes of connection (A and B) is defined as X=(1.ØE1Ø.,Ø.) and placed in the admittance matrix by subroutine ZIADD. Subroutine CXMPY multiplies the scale factor by the admittance and the result is stored in CONST.

Then the nonlinear dependent source data record is rewritten to file 2Ø by subroutine DATAWR and the scale factor is placed in the admittance matrix by subtracting it from the matrix elements (A,XX) and (B,YY) and adding it to the elements (A,YY) and (B,XX).

If neither node A nor XX is ground (A*XX≠Ø), the location of the element (A,XX) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

I= LOCATE(A,XX)= 2*((XX-1)*ORDER+A)-1

where ORDER is the number of elements in each column of the matrix transmitted to subroutine MTNDS through global common. The absolute address is computed by I=DATREC-1+I and the element is read from file 2Ø by subroutine DATARD and stored in PE. The complex scale factor is subtracted from PE by subroutine CXSUB and the updated matrix element is rewritten to file 2Ø by subroutine DATAWR.

If neither node A nor YY is ground (A*YY≠Ø), the location of

the element (A,YY) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

I= LOCATE(A, YY)= 2*((YY-1)*ORDER+A)-1

The absolute address is computed by I=DATREC-1+I and the element is read from file 20 by subroutine DATARD and stored in PE. The complex scale factor is added to PE by subroutine CXADD and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

If neither node B nor XX is ground (B*XX≠0), the location of the element (B,XX) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

I= LOCATE(B,XX)= 2*((XX-1)*ORDER+B)-1

The absolute address is computed by I=DATREC-1+I and the element is read from file 20 by subroutine DATARD and stored in PE. The complex scale factor is added to PE by subroutine CXADD and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

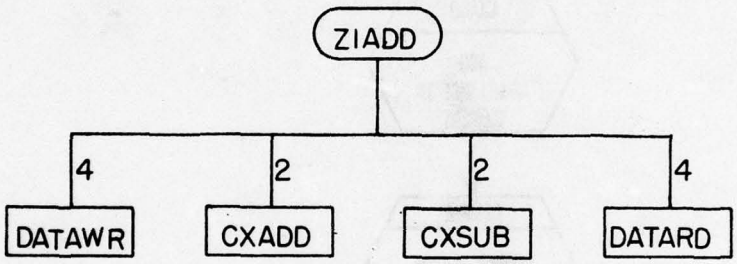
If neither node B nor YY is ground (B*YY≠0), the location of the element (B,YY) relative to the origin of the admittance matrix is calculated by the function LOCATE according to:

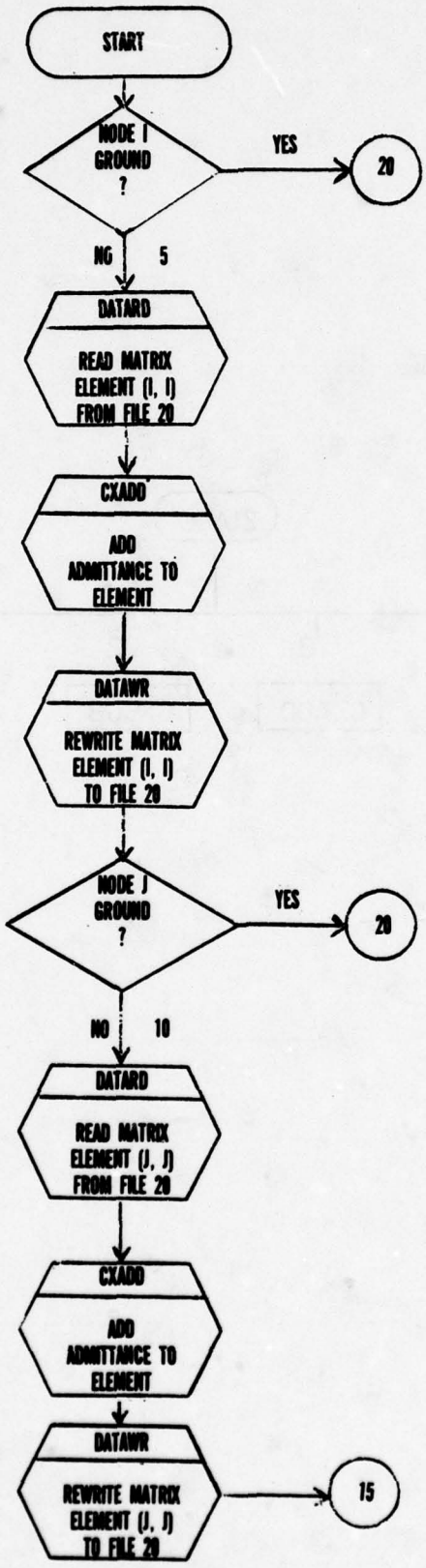
I= LOCATE(B,YY)= 2*((YY-1)*ORDER+B)-1

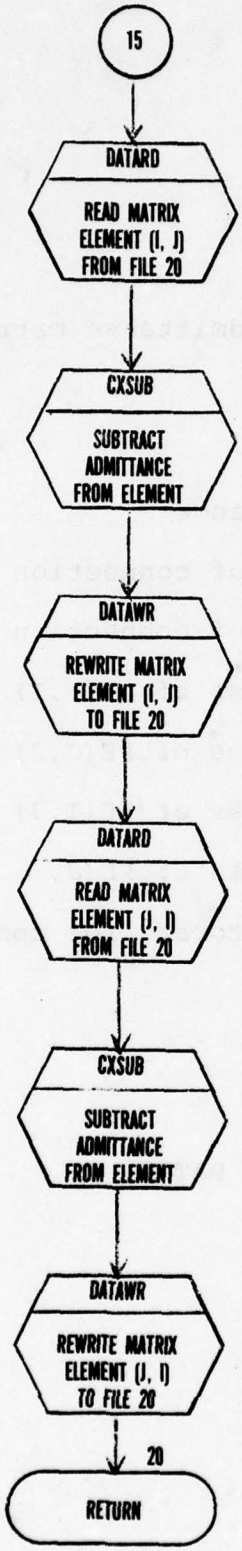
The absolute address is computed by I=DATREC-1+I and the element is read from file 20 by subroutine DATARD and stored in PE. The complex scale factor is subtracted from PE by subroutine CXSUB and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

After the admittances have been properly placed in the

matrix, subroutine MTNDS returns to the calling program.


ZIADD
4
2
2
4
DATAWR
CXADD
CXSUB
DATARD


START
NODE I GROUND ?
YES
20
NO
5
DATARD
READ MATRIX ELEMENT (I, I) FROM FILE 20
CXADD
ADD ADMITTANCE TO ELEMENT
DATAWR
REWRITE MATRIX ELEMENT (I, I) TO FILE 20
NODE J GROUND ?
YES
20
NO
10
DATARD
READ MATRIX ELEMENT (J, J) FROM FILE 20
CXADD
ADD ADMITTANCE TO ELEMENT
DATAWR
REWRITE MATRIX ELEMENT (J, J) TO FILE 20
15


15
DATARD
READ MATRIX ELEMENT (I, J) FROM FILE 20
CXSUB
SUBTRACT ADMITTANCE FROM ELEMENT
DATAWR
REWRITE MATRIX ELEMENT (I, J) TO FILE 20
DATARD
READ MATRIX ELEMENT (J, I) FROM FILE 20
CXSUB
SUBTRACT ADMITTANCE FROM ELEMENT
DATAWR
REWRITE MATRIX ELEMENT (J, I) TO FILE 20
20
RETURN

NAME· ZIADD

TYPE· SUBROUTINE

GENERAL PURPOSE:

Places admittances in admittance matrix

VARIABLES:

A	= Complex admittance
I	= Positive node of connection
J	= Negative node of connection
K	= Relative address of PE(I,I) on disk file 20
L	= Relative address of PE(J,J) on disk file 20
M	= Relative address of PE(I,J) on disk file 20
N	= Relative address of PE(J,I) on disk file 20
PE	= Intermediate storage for complex matrix element

SUBROUTINES CALLED:

CXADD CXSUB DATARD DATAWR

CALLING PROGRAMS:

MTLDS MTNDS PHASE2

DESCRIPTION:

Subroutine ZIADD places the complex admittance for a circuit element in the admittance matrix. Arguments transmitted to the subroutine are the complex admittance A and the numbers of the positive and negative nodes of connection, I and J. No arguments are returned to the calling program.

The admittance is placed in the matrix by adding it to the matrix elements (I,I) and (J,J) and subtracting it from the elements (I,J) and (J,I). If the positive node of connection is ground (I=Ø), subroutine ZIADD returns to the calling program without performing any operations.

Otherwise the location of the element (I,I) relative to the the origin of the admittance matrix is calculated according to:

K= 1+2*(I-1)*(ORDER+1)

where ORDER is the number of elements in each matrix column transmitted to the subroutine in global common. The absolute address is computed as LOC(K)=DATREC-1+K and the element is read from file 20 by subroutine DATARD and stored in PE. The complex admittance A is added to PE by subroutine CXADD and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

If the negative node of connection is ground (J=Ø), subroutine ZIADD returns to the calling program without performing further operations. Otherwise the location of the element (J,J) relative to the origin of the admittance matrix is calculated according to:

I= 1+2*(J-1)*(ORDER+1)

The absolute address is computed as LOC(L)=DATREC-1+L and the element is read from file 20 by subroutine DATARD and stored in PE. The complex admittance A is added to PE by subroutine CXADD and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

Next the location of the element (I,J) relative to the origin of the admittance matrix is calculated according to:

M= K+(J-I)*ORDER*2

The absolute address is computed as LOC(M)=DATREC-1+M and the element is read from file 20 by subroutine DATARD and stored in PE. The complex admittance A is subtracted from PE by subroutine CXSUB and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

Finally the location of the element (J,I) relative to the origin of the admittance matrix is calculated according to:

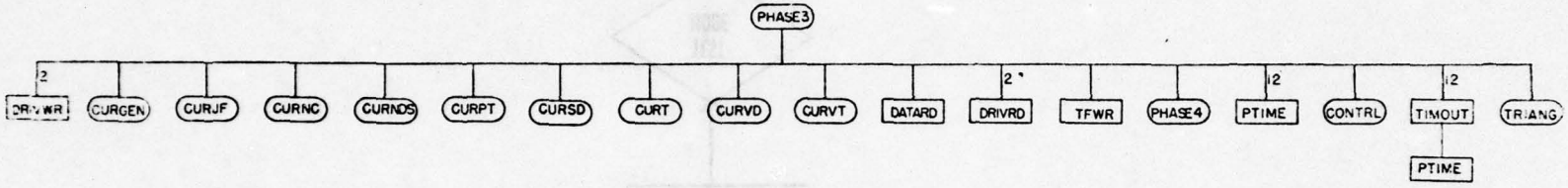
N= K+(J-I)*2

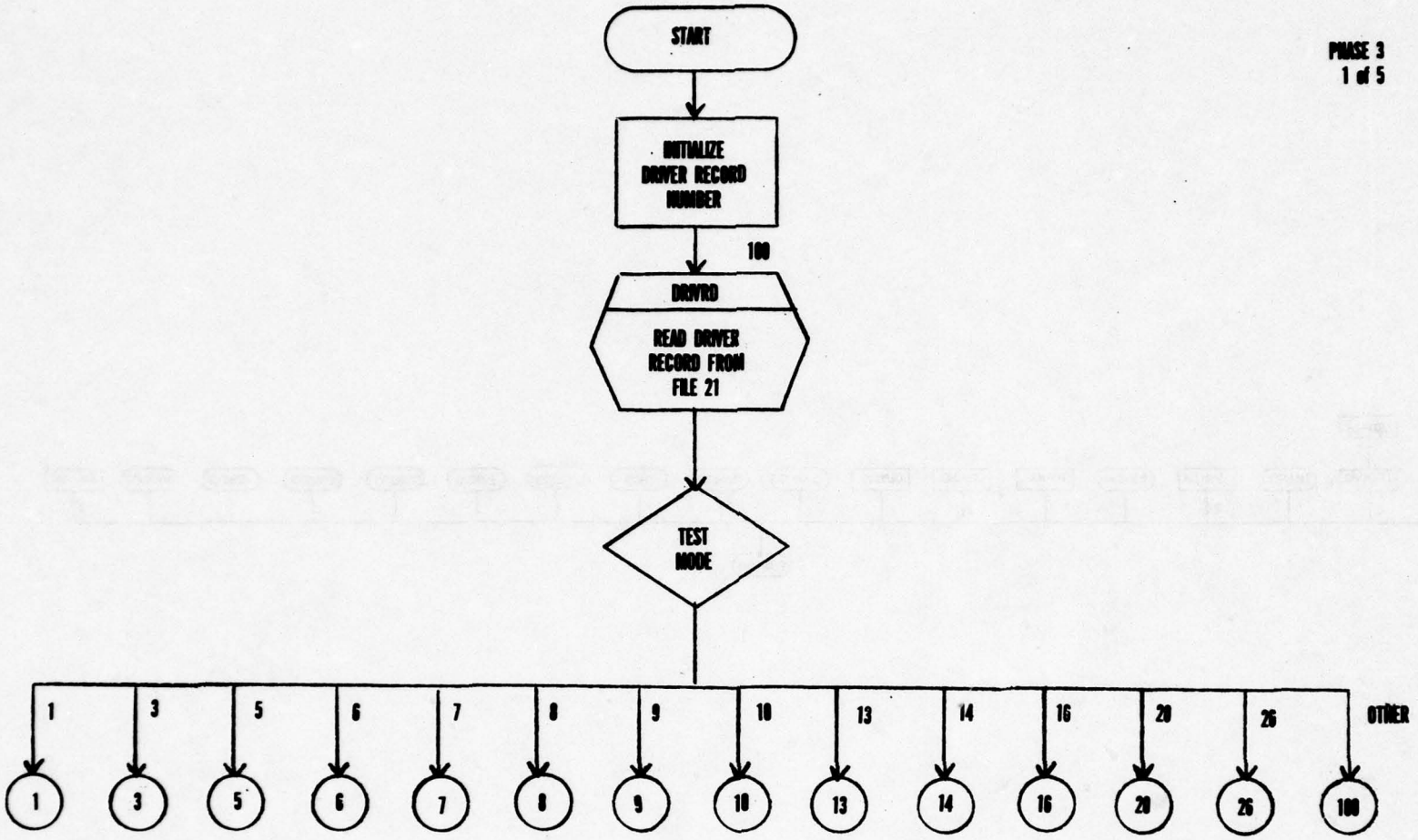
The absolute address is computed as LOC(N)=DATREC-1+N and the element is read from file 20 by subroutine DATARD and stored in PE. The complex admittance A is subtracted from PE by subroutine CXSUB and the updated matrix element is rewritten to file 20 by subroutine DATAWR.

After the admittance has been properly placed in the matrix, subroutine ZIADD returns to the calling program.

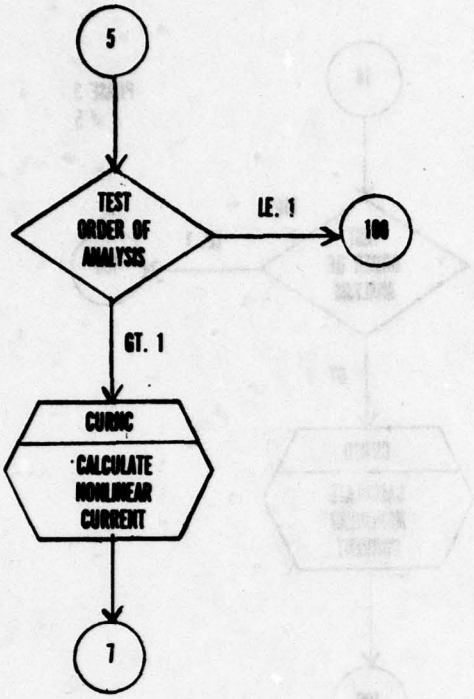
PHASE3

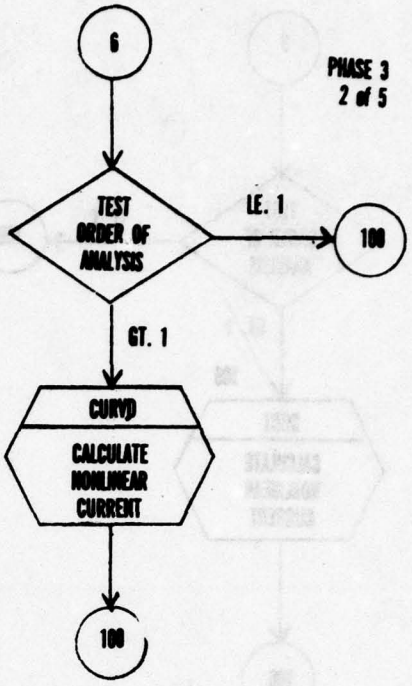
BETA
CONTRL
CROSS
CURGEN
CURJF
CURNC
CURNDS
CURPT
CURSD
CURT
CURVD
CURVT
FNCTN
FRPRM
LOCTF
QFN
SSCODE
TRIANG
UPFRQ
UPSEQ

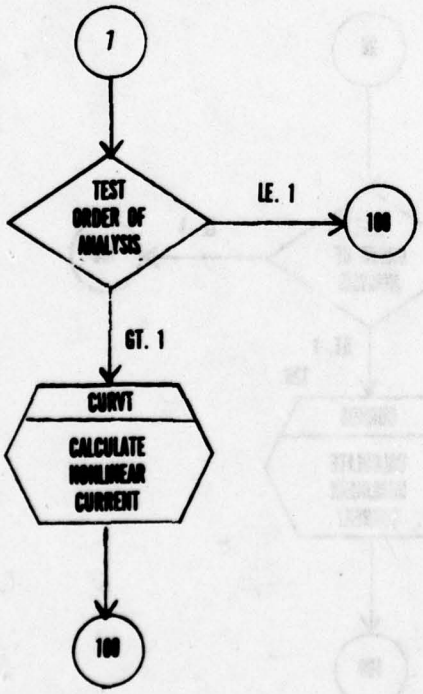

PHASE3
2
DRIVWR
CURGEN
CURJF
CURNC
CURNDS
CURPT
CURSD
CURT
CURVD
CURVT
DATARD
2
DRIVRD
TFWR
PHASE4
12
PTIME
CONTRL
12
TIMOUT
PTIME
TRIANG

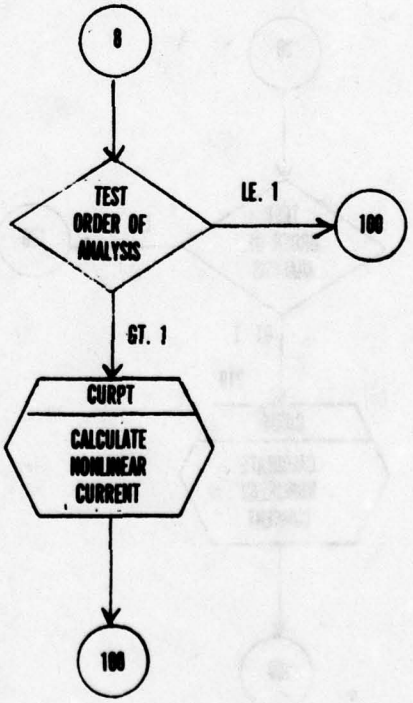

START
INITIALIZE DRIVER RECORD NUMBER
100
DRVRD
READ DRIVER RECORD FROM FILE 21
TEST MODE
1
3
5
6
7
8
9
10
13
14
16
20
26
OTHER
1
3
5
6
7
8
9
10
13
14
16
20
26
100

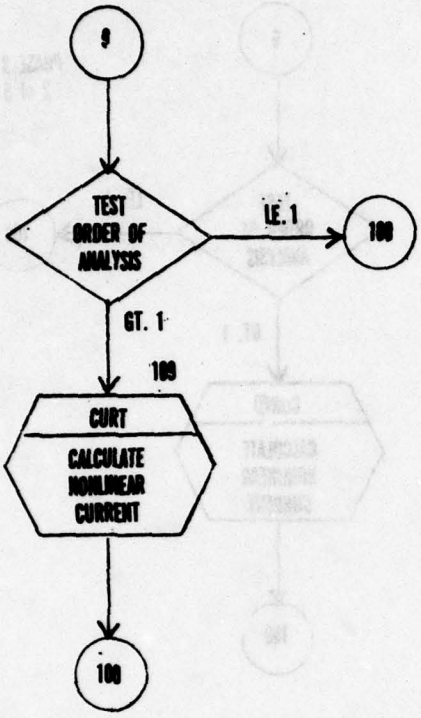
PHASE 3
1 of 5

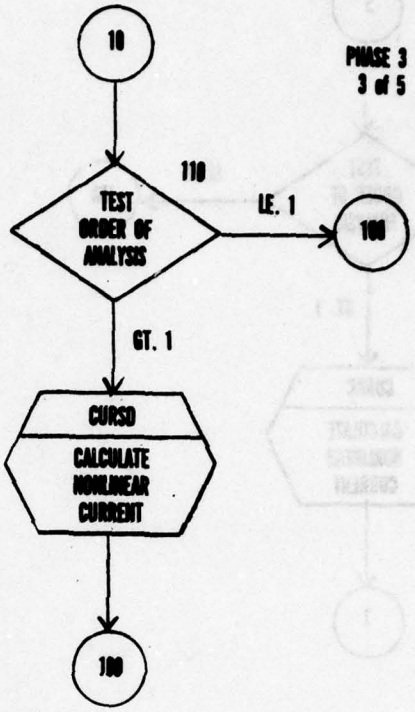

5
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
CURNC
CALCULATE NONLINEAR CURRENT
7

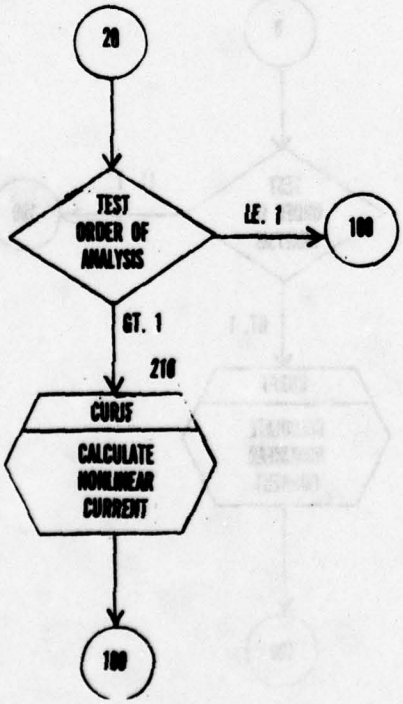

6
PHASE 3
2 of 5
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
CURVD
CALCULATE NONLINEAR CURRENT
100

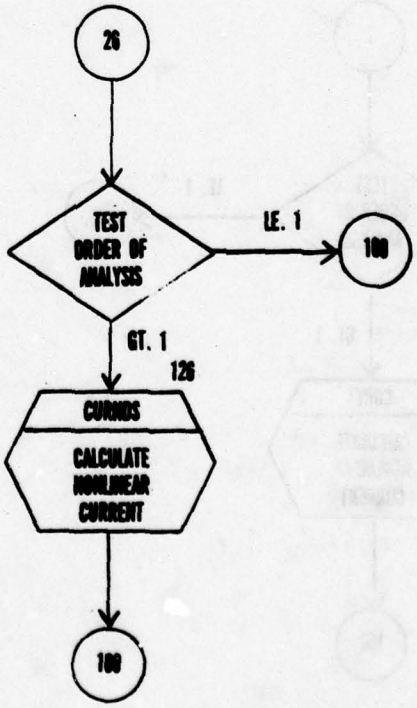

7
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
CURVT
CALCULATE NONLINEAR CURRENT
100

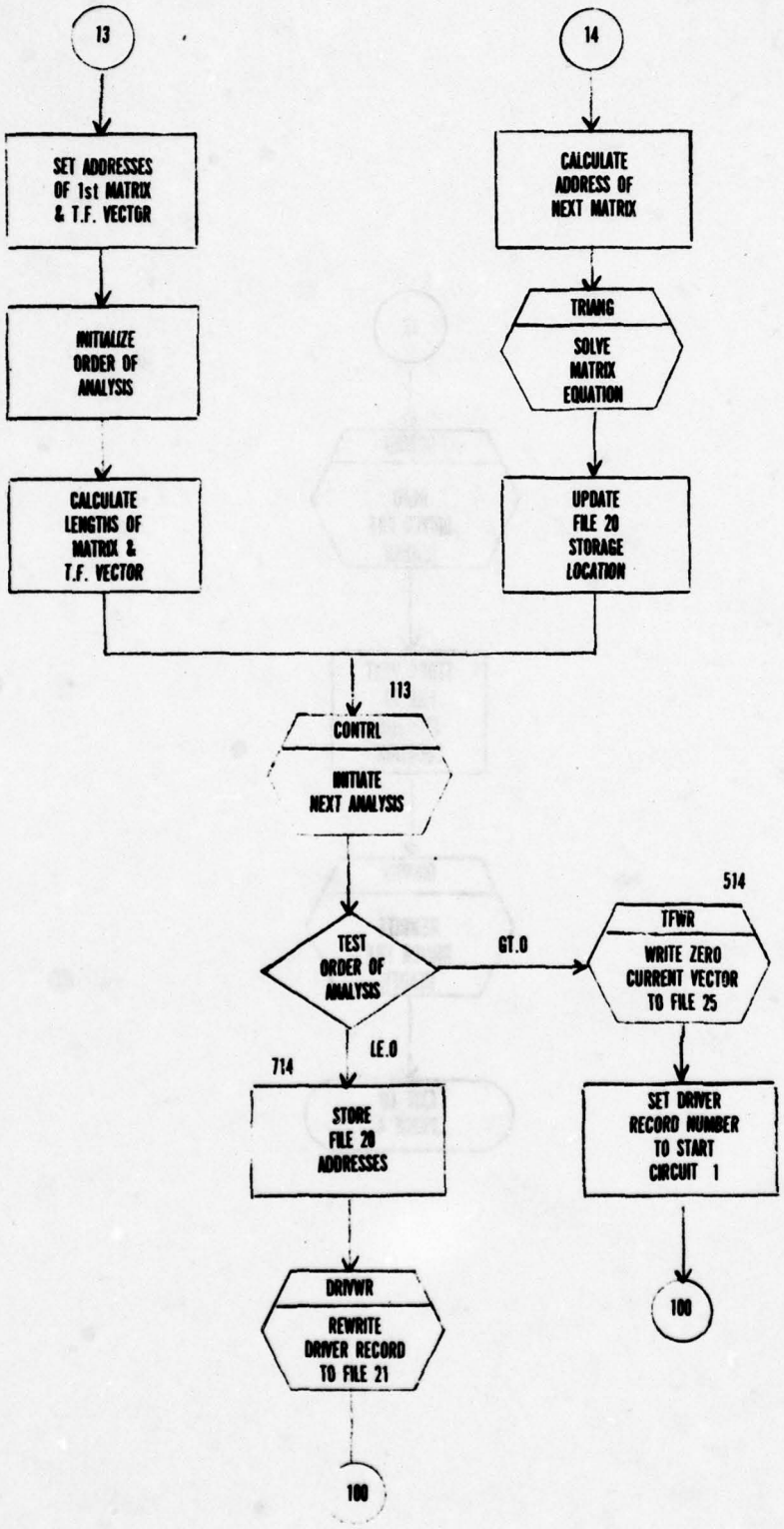

8
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
CURPT
CALCULATE NONLINEAR CURRENT
100


9
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
109
CURT
CALCULATE NONLINEAR CURRENT
100


10
PHASE 3
3 of 5
110
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
CURSD
CALCULATE NONLINEAR CURRENT
100


20
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
210
CURJS
CALCULATE NONLINEAR CURRENT
100


26
TEST ORDER OF ANALYSIS
LE. 1
100
GT. 1
126
CURNDS
CALCULATE NONLINEAR CURRENT
100


13
14
SET ADDRESSES OF 1st MATRIX & T.F. VECTOR
CALCULATE ADDRESS OF NEXT MATRIX
INITIALIZE ORDER OF ANALYSIS
TRIANG
SOLVE MATRIX EQUATION
CALCULATE LENGTHS OF MATRIX & T.F. VECTOR
UPDATE FILE 20 STORAGE LOCATION
113
CONTRL
INITIATE NEXT ANALYSIS
514
TFWR
TEST ORDER OF ANALYSIS
GT.0
WRITE ZERO CURRENT VECTOR TO FILE 25
LE.0
714
STORE FILE 20 ADDRESSES
SET DRIVER RECORD NUMBER TO START CIRCUIT 1
DRIVWR
REWRITE DRIVER RECORD TO FILE 21
100
100


16
DRVRD
READ
DRIVER FILE
HEADER
STORE NEXT
FILE ID
STORAGE
LOCATION
DRVWR
REWRITE
DRIVER FILE
HEADER
EXIT TO
PHASE 4

NAME: PHASE3

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates current vectors. Computes linear and nonlinear transfer functions for each possible frequency combination by solving the matrix equation

(ADMITTANCE MATRIX) X (TRANSFER FUNCTION) = (CURRENT VECTOR)

VARIABLES:

CUR = Intermediate storage for current vector element

FREQ = Frequency values

FRTBL = Frequency number table;

Subscript = Sequential frequency number

Value = User-assigned frequency number

LOCPE = Address of 1st admittance matrix on file 20

LOCT = Address of 1st T.F. vector on file 20

LPE = Length of each admittance matrix in words

LT = Length of each T.F. vector in words

MXORD = Maximum order of analysis

NORDR = Order of analysis

S = Angular frequency

SCODE = Frequency combination code

SCOMB = Sequential frequency combination

SEQ = Frequency combination vector

SUBROUTINES CALLED:

CONTRL	CURGEN	CURJF	CURNC	CURNDS
CURPT	CURSD	CURT	CURVD	CURVT
DATARD	DRIVRD	DRIVWR	PHASE4	PTIME
TFWR	TIMOUT	TRIANG		

CALLING PROGRAMS:

PHASE2

DESCRIPTION:

Subroutine PHASE3 is the principal routine of the last computational phase of the NCAP circuit analysis. It controls the creation of current vectors and calculates and stores the transfer functions for each possible combination of the circuit's input frequencies.

The current vector is a single-dimension array of complex elements, one for each node in the circuit. The nonlinear currents calculated in PHASE3 are stored in the vector so that the current at node I is represented by the Ith complex vector element. The current vector is stored on disk file 25. Every complex vector element occupies two decimal storage locations and is individually addressable.

The transfer functions at each node of the circuit are calculated for every possible combination of the circuit's input frequencies. The transfer functions for a single frequency

combination are stored in a vector of complex elements, one for each node in the circuit, so that the transfer function at node I is represented by the Ith complex vector element. The transfer function vectors are stored on disk file 20. Every complex vector element occupies two decimal storage locations and is individually addressable. Successive transfer function vectors are stored contiguously in the order they are calculated in PHASE3, as one data record associated with the End Circuit driver record.

The construction of current vectors and calculation of transfer functions in subroutine PHASE3 is controlled by scanning the driver file $(2^{MXORD} - 1)$ times, once for each possible combination of the MXORD input frequencies. The Start Circuit driver record initiates the construction of a current vector. During the construction in PHASE3 the current vector remains disk-resident. When a current is to be placed in the vector, the appropriate element is read from file 25, updated, and rewritten to its original location.

As the driver file scan proceeds, the currents generated by circuit elements are calculated and placed in the vector as their driver records are encountered. The End Circuit driver record terminates the current vector construction for a particular frequency combination, calculates and stores the transfer functions for that frequency combination, and restarts the driver file scan for the next combination.

The first driver file scan begins by initializing the driver file record number to DRVREC=1. A driver record is read from

file 21 by subroutine DRIVRD, and subroutine PHASE2 responds to the MODE of the record as follows:

MODE=1 Driver File Header

The next available file 20 data storage location is set by DATREC=MISC(1) and the total number of input frequencies for the circuit is taken as MXORD=MISC(2). The 20-word frequency table is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The frequency values are transferred from BUFF(1)-(10) to the decimal vector FREQ and the frequency numbers are transferred from BUFF(11)-(20) to the integer vector FRTBL. Control then returns to the driver file scan.

MODE=13 Start Circuit

The disk address of the first admittance matrix on file 20 is set to LOCPE=MISC(5), the disk address of the first transfer function vector on file 20 is taken as LOCT=DATREC, and the number of nodes in the circuit is set by ORDER=MISC(4). The order of analysis is initialized to NORDR=0. The length of each transfer function vector (in words) is calculated by LT=2*ORDER and the length of each admittance matrix (in words) is calculated by LPE=LT*ORDER.

Subroutine CONTRL is called to initiate the PHASE3

calculations for the first frequency combination. It updates the order of analysis, forms the frequency combination, and creates and stores a table of selected lower order transfer function vectors for use by the current generator subroutines. The call to subroutine CONTRL and the remainder of the initialization is performed by the END CIRCUIT code, and is described in detail under MODE=14. After the initialization is complete, subroutine PHASE3 resumes the driver file scan.

MODE=3 Generator

For linear frequency combinations (NORDR=1), subroutine CURGEN is called to compute the generator current and place it in the current vector. Control then returns to the driver file scan. For higher orders of analysis (NORDR≠1), subroutine PHASE3 returns to the driver file scan without further processing.

MODE=5 Nonlinear Components

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further processing. For higher orders of analysis (NORDR.GT.1), subroutine CURNC is called to compute the nonlinear currents and place them in the current vector. Control then returns to the driver file scan.

MODE=6 Vacuum Diode

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further processing. For higher orders of analysis (NORDR.GT.1), subroutine CURVD is called to compute the nonlinear current and place it in the current vector. Control then returns to the driver file scan.

MODE=7 Vacuum Triode

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further processing. For higher orders of analysis (NORDR.GT.1), subroutine CURVT is called to compute the nonlinear current and place it in the current vector. Control then returns to the driver file scan.

MODE=8 Vacuum Pentode

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further

processing. For higher orders of analysis (NORDR.GT.1), subroutine CURPT is called to compute the nonlinear currents and place them in the current vector. Control then returns to the driver file scan.

MODE=9 Bipolar Junction Transistor

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further processing. For higher orders of analysis (NORDR.GT.1), subroutine CURT is called to compute the nonlinear currents and place them in the current vector. Control then returns to the driver file scan.

MODE=10 Semiconductor Diode

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further processing. For higher orders of analysis (NORDR.GT.1), subroutine CURSD is called to compute the nonlinear current and place it in the current vector. Control then returns to the driver file scan.

MODE=20 Junction Field Effect Transistor

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further processing. For higher orders of analysis (NORDR.GT.1), subroutine CURJF is called to compute the nonlinear currents and place them in the current vector. Control then returns to the driver file scan.

MODE=26 Nonlinear Dependent Source

For linear frequency combinations (NORDR=1), subroutine PHASE3 returns to the driver file scan without further processing. For higher orders of analysis (NORDR.GT.1), subroutine CURNDS is called to compute the nonlinear current and place it in the current vector. Control then returns to the driver file scan.

MODE=14 End Circuit

The transfer function vector is calculated by subroutine TRIANC and stored on file 20. The next available file 20 storage location is updated to DATREC=DATREC+LT and subroutine CONTRL is called to initiate the calculations for the next frequency combination. It updates the order of analysis, forms the next frequency combination, and creates and stores a table of selected

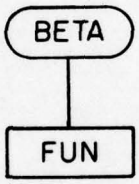
lower order transfer function vectors for use by the current generator subroutines. After returning from subroutine CONTRL, if the last transfer function vector has not been computed (NORDR.GT.0), the current vector storage area is cleared, one element at a time as follows: subroutine TFWR writes the complex number CUR=(0.,0.) to the first LT/2 vector elements on file 25. The driver file record number is reset for the next driver file scan by DRVREC=MISC(1)+1 and control returns to the driver file scan.

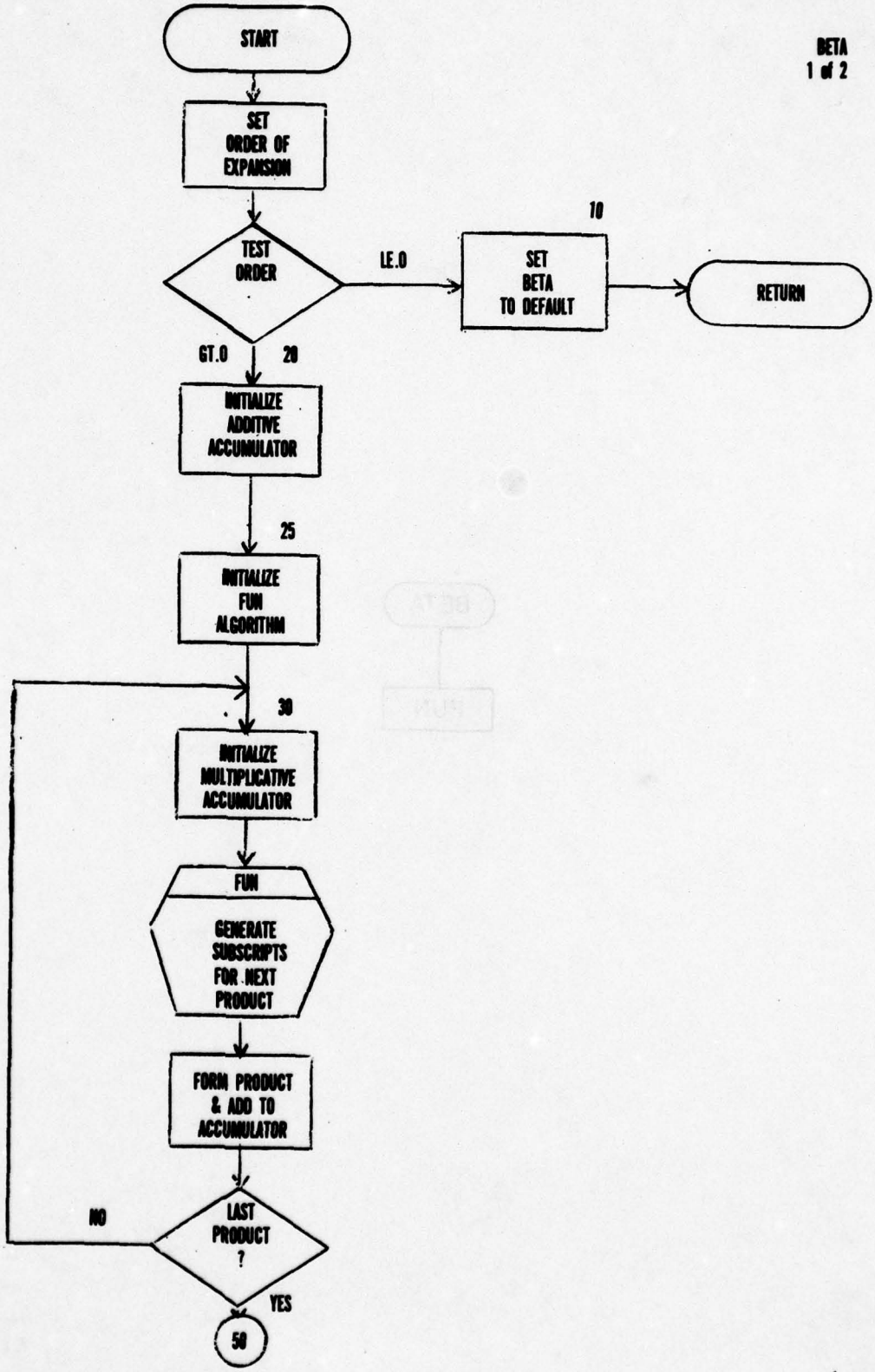
If the last transfer function vector has been calculated (NORDP=-1), the transfer function data record is attached to the End Circuit driver record by MISC(2)=LOCT. The next available file 20 storage location is saved in MISC(3) and the End Circuit record is rewritten to the driver file by subroutine DRIVWR. Control then returns to the driver file scan.

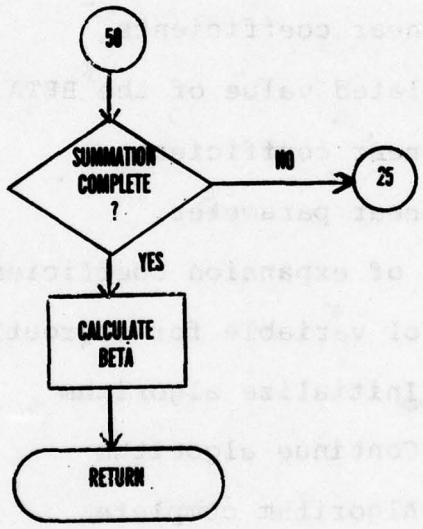
MODE=16 End

The driver file header is read by subroutine DRIVRD, updated to reflect the next available file 20 storage location, and rewritten by subroutine DRIVWR. Program control then transfers to subroutine PHASE4.

For other values of MODE, control returns to the driver file scan without further processing by subroutine PHASE3.


BETA
FUN


START
BETA
1 of 2
SET
ORDER OF
EXPANSION
TEST
ORDER
LE.0
10
SET
BETA
TO DEFAULT
RETURN
GT.0
20
INITIALIZE
ADDITIVE
ACCUMULATOR
25
INITIALIZE
FUN
ALGORITHM
30
INITIALIZE
MULTIPLICATIVE
ACCUMULATOR
FUN
GENERATE
SUBSCRIPTS
FOR NEXT
PRODUCT
FORM PRODUCT
& ADD TO
ACCUMULATOR
LAST
PRODUCT
?
NO
YES
50


50
SUMMATION COMPLETE ?
NO
25
YES
CALCULATE BETA
RETURN

NAME: BETA

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates the function B_M^K for use by subroutine CURT

VARIABLES:

ALPHS = Nonlinear coefficients

B = Calculated value of the BETA function

HM = Nonlinear coefficients

ICHMO = Nonlinear parameter

II = Order of expansion coefficient

ISW = Control variable for subroutine FUN

-1 = Initialize algorithm

Ø = Continue algorithm

1 = Algorithm complete

K = Index for outer summation of current calculation

KI = Subscripts calculated in subroutine FUN

KS = Nonlinear coefficients

M = Index for inner summation of current calculation

PAK = Additive accumulator

PK = Multiplicative accumulator

SNCS = Nonlinear parameter

SUBROUTINES CALLED:

FUN

CALLING PROGRAMS:

CURT

DESCRIPTION:

Subroutine BETA calculates the function B_M^K for use in calculating the nonlinear current for the bipolar junction transistor. Arguments transmitted to the subroutine are: K, the index for the inner summation in the nonlinear current calculation; M, the index for the outer summation in the nonlinear current calculation; ALPHS, HM, and KS, nonlinear coefficients calculated in Phase 1; ICHMØ and SNCS, nonlinear parameters calculated in Phase 1. The calculated function B_M^K is returned to the calling program in B.

If K=M, the function B_M^K is calculated according to B=ICHMØ*HM(M) and subroutine BETA returns to the calling program. Otherwise the function B_M^K is defined as:

$$B = SNCS * HM(M) * \sum_{I=1}^{K-M} ALPHS(I) * F_I^{K-M}(KS)$$

where each $F_I^{K-M}(KS)$, a sum of products of nonlinear coefficients

KS, is computed by repeated calls to subroutine FUN under the control of the variable ISW. For ISW=-1 the algorithm of subroutine FUN is initialized and each call returns a set of integers in the vector KI which constitute the subscripts of KS's to be multiplied together to form one additive term of the expression. Upon return from subroutine FUN, if ISW=0 additional products remain to be calculated, and subroutine FUN is called again. When the expression is complete, SWTCH is returned as 1 and the algorithm stops.

The complete procedure for calculating the function B_M^K in subroutine BETA is summarized as follows:

STEP 1: Initialization

a) Clear additive accumulator PAK=0.

b) Initialize index for summation I=1

c) Initialize FUN algorithm by ISW=-1

STEP 2: Calculate One Additive Term ALPHS(I)*F_I^{K-M}(KS)

a) Initialize multiplicative accumulator PK=1.

b) Call subroutine FUN to generate subscripts KI(J), J=1,2,...I for one additive term.

c) Form the product

$$PK = \prod_{J=1}^{I} KS(KI(J))$$

d) Form product ALPHS(I)*PK and add to accumulator PAK.

STEP 3· Test ISW

a) If ISW=0, more products remain to be calculated. Go to Step 2.

b) Otherwise one additive term complete in PK, go to Step 4.

STEP 4: Update and Test Summation Index I

a) Increment I=I+1

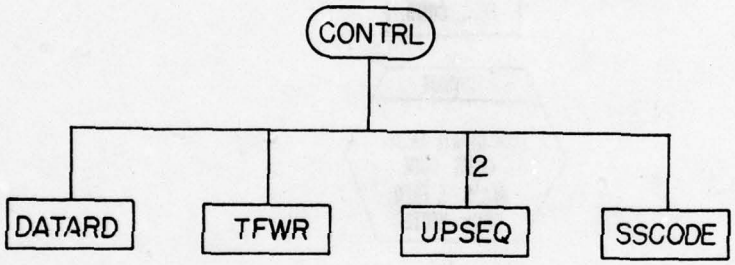
b) If (I.LE.(K-M)), go to Step 1c.

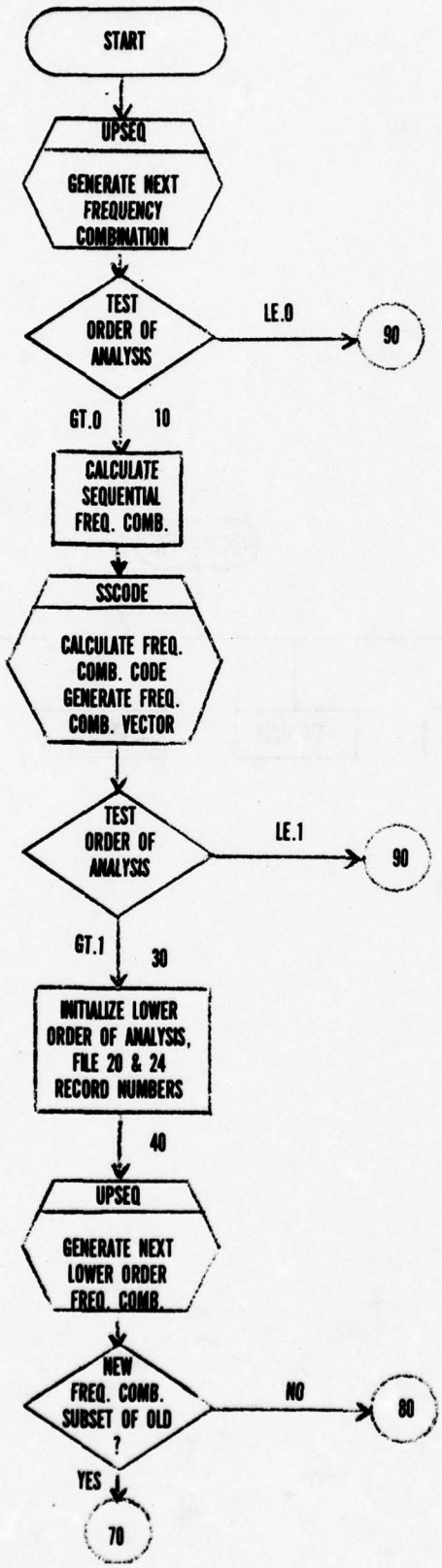
c) Otherwise summation complete in PAK, go to Step 5.

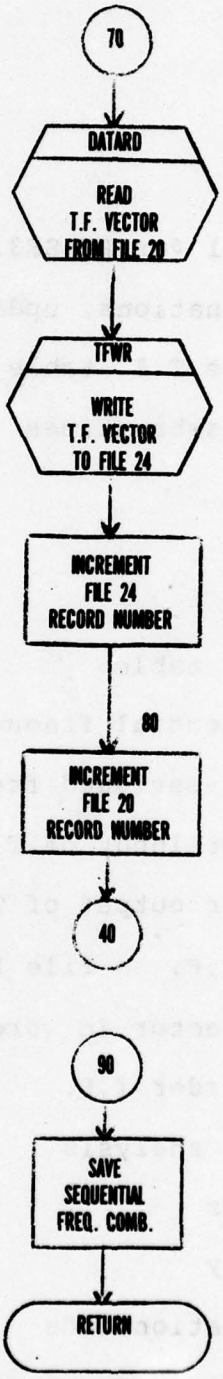
STEP 5· Complete Calculation of B_M^K

a) Compute B=SNCS*HM(M)*PAK.

b) Return to calling program with B_M^K in B.


CONTRL
2
DATARD
TFWR
UPSEQ
SSCODE


START
UPSEQ
GENERATE NEXT FREQUENCY COMBINATION
TEST ORDER OF ANALYSIS
LE.0
90
GT.0
10
CALCULATE SEQUENTIAL FREQ. COMB.
SSCODE
CALCULATE FREQ. COMB. CODE GENERATE FREQ. COMB. VECTOR
TEST ORDER OF ANALYSIS
LE.1
90
GT.1
30
INITIALIZE LOWER ORDER OF ANALYSIS, FILE 20 & 24 RECORD NUMBERS
40
UPSEQ
GENERATE NEXT LOWER ORDER FREQ. COMB.
NEW FREQ. COMB. SUBSET OF OLD ?
NO
80
YES
70


70
DATARD
READ
T.F. VECTOR
FROM FILE 20
TFWR
WRITE
T.F. VECTOR
TO FILE 24
INCREMENT
FILE 24
RECORD NUMBER
80
INCREMENT
FILE 20
RECORD NUMBER
40
90
SAVE
SEQUENTIAL
FREQ. COMB.
RETURN

NAME: CONTRL

TYPE: SUBROUTINE

GENERAL PURPOSE:

Provides frequency control for PHASE3.
Cenerates frequency combinations, updates
order of analysis, creates T.F. table for
use by current generator subroutines

VARIAPLES:

FREQ = Frecuency values

FRTBL = Frequency number table:

Subscript = Sequential frequency number

Value = User-assigned frequency number

K = Record number for input of T.F. from file 20

LL = Record number for output of T.F. to file 24

LOCT = Address of 1st T.F. on file 20

LT = Length of T.F. vector in words

MORDR = Order of lower-order T.F.

MXORD = Maximum order of analysis

NORDR = Order of analysis

S = Angular frecuency

SCODE = Frecuency combination code

SCOMB = Secuential frecuency combination

SEQ = Frecuency combination vector

SEQ1 = Lower-order frequency combination vector

T = Intermediate storage for T.F. element

SUBROUTINES CALLED:

DATARD SSCODE TFWR UPSEQ

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CONTRL initiates the PHASE3 calculations of the current vector and transfer functions for a particular frequency combination by forming that frequency combination and creating a table of selected lower order transfer function vectors from which the circuit's nonlinear currents are derived.

Arguments transmitted to CONTRL from subroutine PHASE3 are: MXORD, the number of input frequencies for the circuit (maximum order of analysis); FREQ, the input frequency values; FRTBL, the frequency number table; LOCT, the disk address of the first transfer function vector on file 20; LT, the length of each transfer function vector in words; NORDR, the number of frequencies in the previous frequency combination (order of analysis); and SEQ, the previous frequency combination vector. Arguments returned to subroutine PHASE3 are: SCODE, the new frequency combination code; NOPDR, the number of frequencies in the new frequency combination (updated order of analysis); S, the calculated angular frequency for the new frequency

combination and SEQ, the new frequency combination vector.

Upon entering subroutine CONTRL, the next frequency combination is generated by subroutine UPSEQ. If the last possible frequency combination has already been generated (NORDR.LE.0), subroutine CONTRL returns to the calling program without further processing. Otherwise NORDR (the updated order of analysis) represents the number of frequencies in the new combination which is transmitted to subroutine CONTRL in the integer vector SEQ, where the NORDR frequencies in the combination are stored according to increasing sequential frequency numbers in the first NORDR elements of SEQ.

Next the sequential frequency combination is derived from the frequency combination vector according to:

$$SCOMB = \sum_{I=1}^{NORDR} SEQ(I)$$

Subroutine SSCODE is called to calculate the frequency combination code, SCODE, and the angular frequency, S.

Because linear currents are not derived from previous transfer functions, no transfer function vector table is created for first order frequency combinations. Therefore for NOPDR=1, subroutine CONTRL stores the sequential frequency combination in SEQ(11) and returns to the calling program without further processing. For higher orders of analysis (NCRDR.GT.1), the remainder of subroutine CONTRL is devoted to creating and storing

the transfer function vector table for the new frequency combination.

In the NCAP circuit analysis, the nonlinear currents for a particular frequency combination are derived from previously calculated transfer functions. In particular, an nth order nonlinear current evaluated at the frequency combination (f1,f2 ,....fn) is derived from the 2^n -2 lower order transfer function vectors from frequency combinations which are subsets of (f1,f2,...fn). For example in a circuit with four input frequencies, the third order nonlinear currents at the frequency combination (f1,f3,f4) are derived from the 2^3 -2=6 transfer function vectors from all the first and second order combinations of f1, f3, and f4:

T(f1)

T(f3)

T(f4)

T(f1,f3)

T(f1,f4)

T(f3,f4)

The transfer functions vectors T(f2),T(f1,f2), T(f2,f3), T(f2, f4) do not contribute to the nonlinear current since they correspond to frequency combinations that are not subsets of (f1,f3,f4).

In subroutine CONTRL the lower-order transfer function vectors required for the nonlinear current calculations at the new frequency combination are selected from file 20 and placed in a table on file 24 for use by the nonlinear current generator

subroutines of Phase 3. The selection process involves recreating every lower-order frequency combination while maintaining the file 20 address of the corresponding transfer function vector. When a subset of the new frequency combination is encountered, the appropriate transfer function vector is read from file 20 and appended to the table on file 24. This procedure guarantees that only those transfer function vectors required for the new frequency combination are included in the table.

The lower order frequency combinations are generated by repeated calls to subroutine UPSEQ. The order of the combination is maintained in MORDR and each frequency combination is returned to subroutine CONTRL in the first MORDR elements of the integer vector SEQ1.

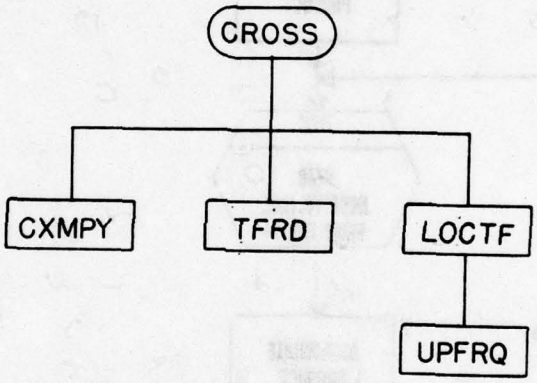
The creation of the transfer function vector table in subroutine CONTRL begins by initializing MORDR=0. The file 20 address is initialized to K=LOCT and the file 24 address is set to LL=1. Subroutine UPSEQ is called to generate a frequency combination of order MORDR, which is returned in SEQ1.

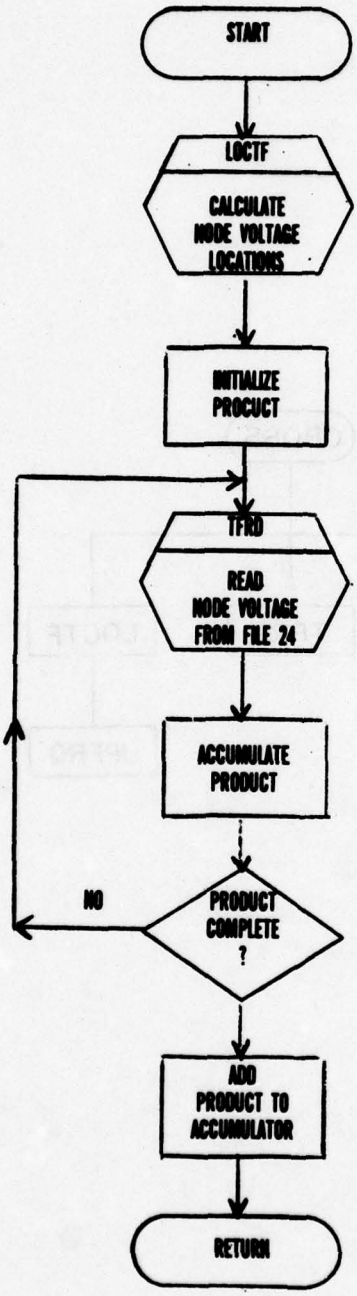
If the last combination has not been generated (MORDR.LT NORDR), subroutine CONTRL determines whether the frequency combination in SEQ1 is a subset of the combination in SEQ. If every SEQ1(I), I=1,MORDR equals some SEQ(J), J=1,NORDR, then the corresponding transfer function vector stored at K is read from file 20 by subroutine DATARD and appended to the table on file 24 by subroutine TFWR. Then the file 24 address is updated to LL=LL+LT, the file 20 address is updated to K=K+LT, and the next

frequency combination is generated.

If SEQ1 is not a subset of SEQ, the read/write operations are by-passed and only the file 20 address is updated before generating the next frequency combination.

This process continues until MORDR=NORDR, indicating that all lower order combinations have been generated and the transfer function vector table is complete. Subroutine CONTRL then saves the sequential frequency combination in SEQ(11) and returns to the calling program.


CROSS
CXMPY
TFRD
LOCTF
UPFRQ


START
LOCTF
CALCULATE
NODE VOLTAGE
LOCATIONS
INITIALIZE
PROCUCT
TFRD
READ
NODE VOLTAGE
FROM FILE 24
ACCUMULATE
PRODUCT
NO
PRODUCT
COMPLETE
?
ADD
PRODUCT TO
ACCUMULATOR
RETURN

NAME: CROSS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Accesses voltages in T.F. vector table, computes T.F. product for one frequency permutation, adds product to accumulator for symmetrization by subroutine FNCTN

VARIABLES:

FNK	=	Subscripts calculated in subroutine FUN
FREQC	=	Permuted frequency vector
II	=	Subscript for permuted frequency vector
K	=	Number of voltages in T.F. product
LCCTF	=	Origin of T.F. vector on file 24
N	=	Order of nonlinear current being calculated
NODE	=	Base node of circuit element
Q	=	Node pattern of T.F. product
SUBS	=	Locations of voltages in T.F. vector
TS	=	Intermediate storage for voltage
YY	=	Sum of T.F. products
ZZ	=	T.F. product

SUBROUTINES CALLED:

CXMPY LOCTF TFRD

CALLING PROGRAMS:

CURPT FNCTN

DESCRIPTION:

Subroutine CROSS accesses node voltages in the modified transfer function table, computes the voltage product for a single frequency permutation, and adds the product to an accumulator for use by subroutine FNCTN. Arguments transmitted to the subroutine are: FNK, the orders of the voltages in the transfer function product· Q, the node numbers (relative to the base node of the device) of the voltages in the transfer function product: FREQC, the frequency permutation vector: N, the order of the nonlinear current being calculated; K, the number of voltages in the transfer function product: and YY, the accumulated sum of transfer function products. The updated sum YY is returned as an argument to the calling program.

Subroutine CROSS begins by calculating the storage locations for each of the K voltages in the transfer function product, saving the locations in SUBS(I) for I=1,2,...K. The voltages are stored in the modified transfer function table on file 24. For an Nth order nonlinear curent calculation, the table contains 2^N -2 transfer function vectors. Each vector occupies one record and contains ORDER complex elements. The location of a particular node voltage in the transfer function table is derived by:

VOLTAGE LOCATION = T.F. ORIGIN + NODE DISPLACEMENT

First the function subprogram LOCTF calculates the record number of the transfer function vector containing the desired voltage. Then the transfer function origin is defined as:

TRANSFER FUNCTION ORIGIN = (LOCTF-1)*ORDER

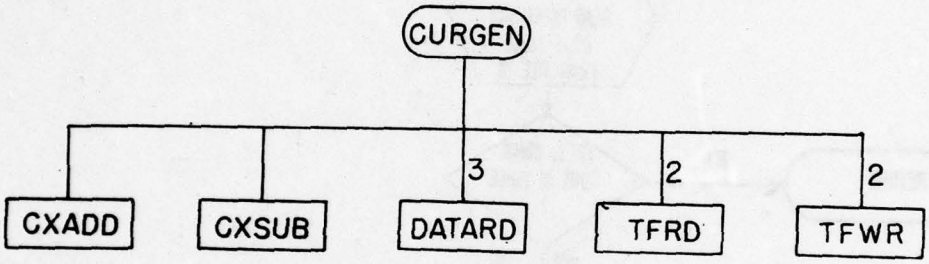
Then using Q(I), the location of the Ith node voltage relative to the base node of the circuit element, and NODE, the number of the base node, the node displacement is defined as:

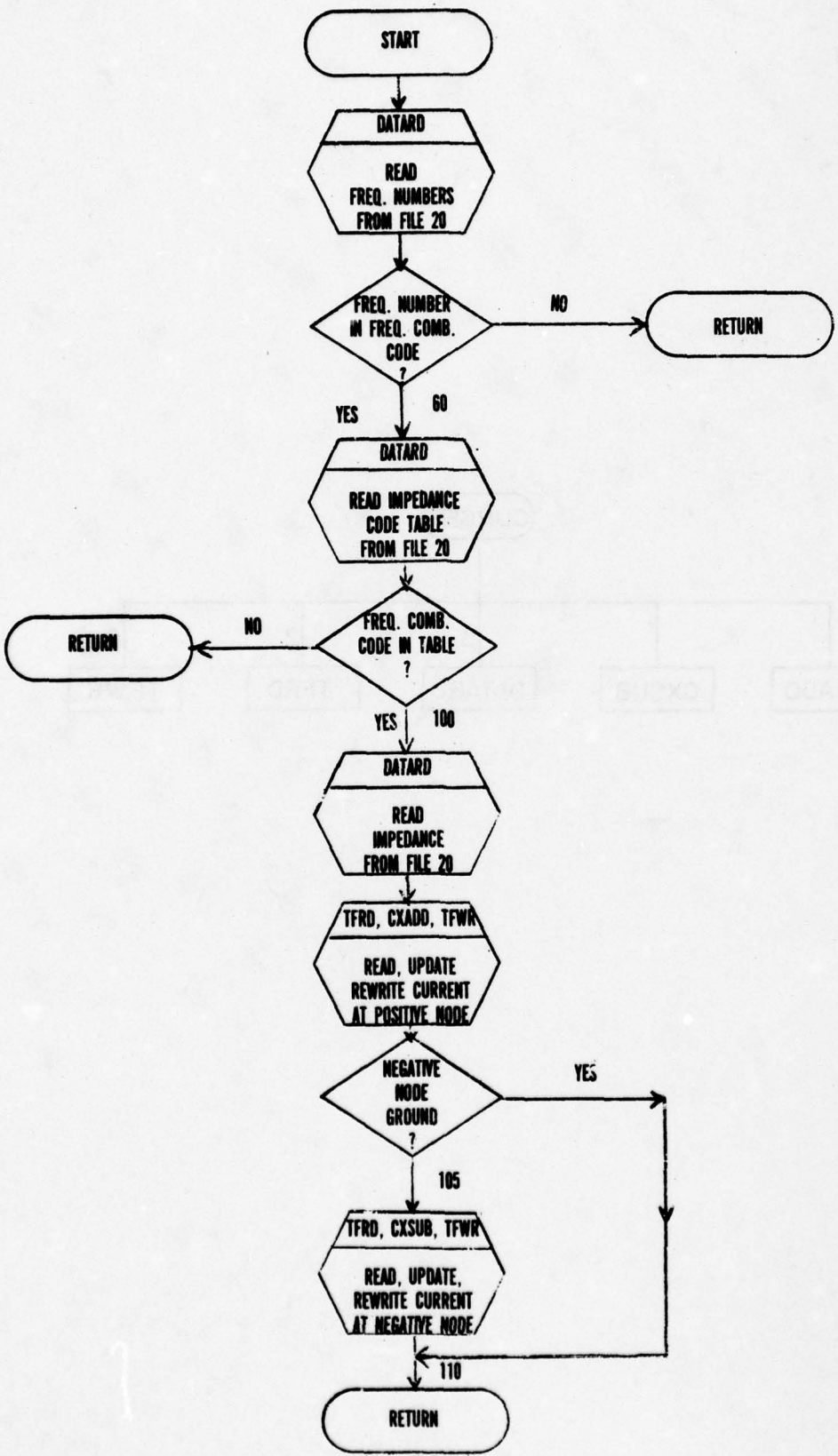
NODE DISPLACEMENT = Q(I)+NODE-1

The complete expression for the location of the Ith voltage in the transfer function product is:

SUBS(I) = ((LOCTF-1)*ORDER)+Q(I)+NODE-1

After all of the K voltage locations have been calculated, the transfer function product is formed. A voltage is read from file 24 by subroutine TFRD and stored in TS. The product is accumulated in ZZ by subroutine CXMPY. When all K voltages have been accessed and multiplied, the resulting product is added to the accumulator YY and subroutine CROSS returns to the calling program.


CURGEN
CXADD
CXSUB
3
DATARD
2
TFRD
2
TFWR


START
DATARD
READ FREQ. NUMBERS FROM FILE 20
FREQ. NUMBER IN FREQ. COMB. CODE ?
NO
RETURN
YES
GO
DATARD
READ IMPEDANCE CODE TABLE FROM FILE 20
FREQ. COMB. CODE IN TABLE ?
NO
RETURN
YES
100
DATARD
READ IMPEDANCE FROM FILE 20
TFRD, CXADD, TFWR
READ, UPDATE REWRITE CURRENT AT POSITIVE NODE
NEGATIVE NODE GROUND ?
YES
105
TFRD, CXSUB, TFWR
READ, UPDATE, REWRITE CURRENT AT NEGATIVE NODE
110
RETURN

NAME: CURGEN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates linear current for generator and places it in current vector

VARIABLES:

A = Linear current (impedance)

CUR = Current vector element

FNC = User-assigned frequency numbers

I = Relative location of impedance on file 20

IMTPL = Impedance code table

NFREQ = Number of frequencies

NIMPS = Number of impedances

NODE = Positive node of connection

NODEB = Negative node of connection

SCODE = Frequency combination code

SUBROUTINES CALLED:

CXADD CXSUB DATARD TFRD TFWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

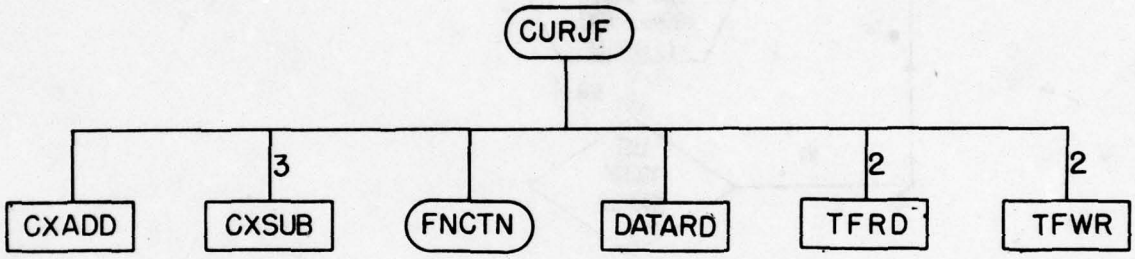
Subroutine CURGEN calculates the current for the generator at a given frequency combination and places it in the current vector. The frequency combination code SCODE, for which the current is to be calculated, is transmitted to subroutine CURGEN as an argument. The generator is an independent voltage source which generates linear current defined as the generator's impedance at the given frequency combination. The impedances are stored in the generator data record together with a table of corresponding frequency combination codes IMTBL where IMTBL(I) is the frequency combination for the Ith impedance in the data record. The number of input frequencies and impedances for the generator, NFREQ and NIMPS, and the nodes of connection, NODE and NODEB, are stored in the generator driver record in global common.

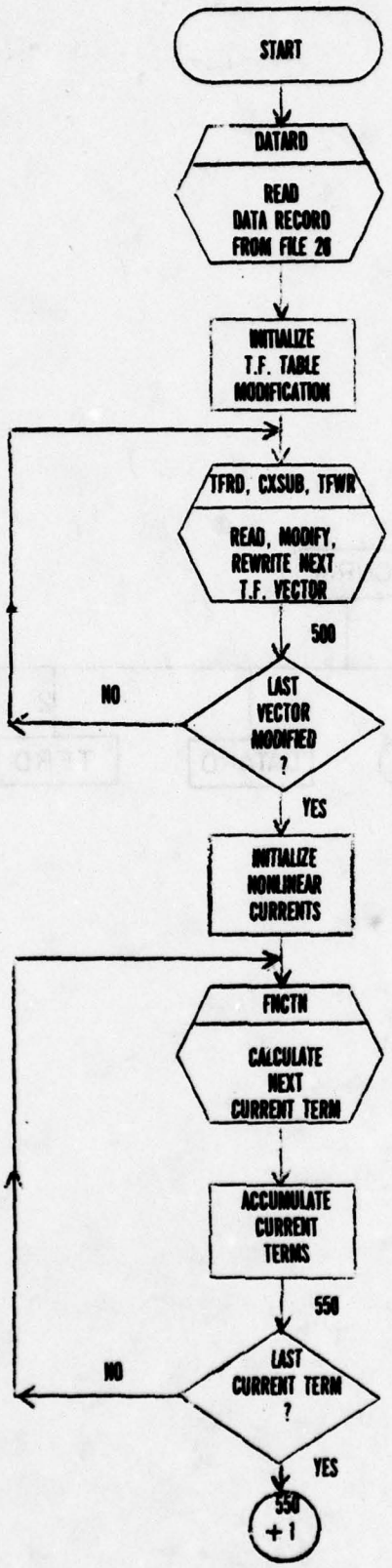
First the user-assigned frequency numbers are read from the generator data record on file 20 by subroutine DATARD and stored in the integer vector FNO. If the frequency number encoded in SCODE is not among the frequencies FNO defined for the generator, subroutine CURGEN returns to the calling program without further processing.

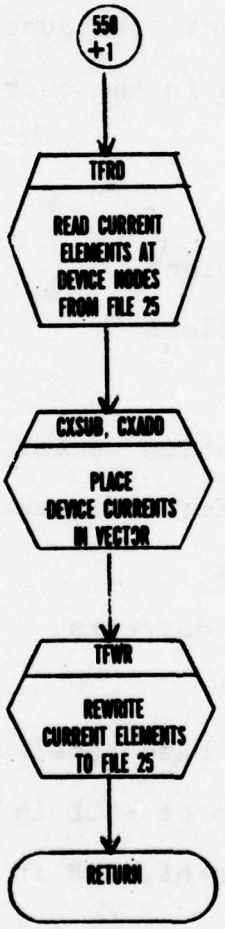
Otherwise the table of frequency combination codes is read from the generator data record on file 20 by subroutine DATARD and stored in IMTBL. Then SCODE is compared against every frequency combination code in the table, IMTBL(I), I=1,2,...NIMPS. If no match is found, subroutine CURGEN returns to the calling program. If a match is found, the table position

I is used to locate the desired impedance in the data record. The impedance is then read from file 20 by subroutine DATARD and stored in A.

The current is placed in the current vector on file 25 by adding A to the current at the positive node of connection, NODE, and subtracting A from the current at the negative node of connection, NODEB. The NODE current, located at I=2*NODE-1, is read from file 25 by subroutine TFRD and stored in CUR. A is added to CUR by subroutine CXADD and the updated current is rewritten to its original location by subroutine FTWR. If the negative node of connection is ground (NODEB=0), subroutine CURGEN returns to the calling program. Otherwise the NODEB current, located at I=2*NODEB-1, is read from file 25 by subroutine TFRD and stored in CUR. A is subtracted from CUR by subroutine CXSUB, the updated current is rewritten to its original location, and subroutine CURGEN returns to the calling program.


CURJF
CXADD
3
CXSUB
FNCTN
DATARD
2
TFRD
2
TFWR


START
DATARD
READ
DATA RECORD
FROM FILE 20
INITIALIZE
T.F. TABLE
MODIFICATION
TFRD, CXSUB, TFWR
READ, MODIFY,
REWRITE NEXT
T.F. VECTOR
500
LAST
VECTOR
MODIFIED
?
NO
YES
INITIALIZE
NONLINEAR
CURRENTS
FNCTN
CALCULATE
NEXT
CURRENT TERM
ACCUMULATE
CURRENT
TERMS
550
LAST
CURRENT TERM
?
NO
YES
550
+1


550
+1
TFRD
READ CURRENT ELEMENTS AT DEVICE NODES FROM FILE 25
CXSUB, CXADD
PLACE DEVICE CURRENTS IN VECTOR
TFWR
REWRITE CURRENT ELEMENTS TO FILE 25
RETURN

NAME: CURJF

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear currents for Junction Field Effect Transistor and places them in the current vector

VARIABLES:

A	= Nonlinear coefficient
C	= Nonlinear coefficient
CGD	= Input value
FREQ	= Frequency combination vector
GG	= T.F. expression for nonlinear current
HG	= Nonlinear current
HH	= Sum of nonlinear currents
HL	= Nonlinear current
I	= Record number of base node voltage (current)
I1	= Relative location of NODE in first T.F.
I2	= Relative location of NODE in last T.F.
I3	= Location of work area on file 24
K	= Index for summation of current calculation
L	= Record number for file 24 work area
LT	= Length of each T.F. vector in words
NODE	= Base node of connection
NORDR	= Order of analysis

S = Angular frequency

X = Base node voltage (current)

X1 = (Base+1) node voltage (current)

X2 = (Base+2) node voltage (current)

SUBROUTINES CALLED:

CXADD CXSUB DATARD FNCTN

TFRD TFWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURJF calculates the nonlinear currents for the junction field effect transistor and places them in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear currents to be calculated and S, the angular frequency. The nonlinear currents are functions of the device coefficients calculated in Phase 1 and the node voltages derived from the lower order transfer functions stored on file 24. The base node of connection, NODE, is stored in the JFET driver record in global common. The remaining nodes are numbered sequentially by the system as NODE+1, NODE+2, and NODE+3.

First the JFET data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The node voltages for the nonlinear current calculations are computed by replacing the transfer function at NODE+2 with the difference

between the transfer functions at NODE and NODE+2 for every vector in the table. This "modified transfer function table" serves as the basis for the nonlinear current calculations.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the circuit and occupies two decimal storage locations. The transfer functions are stored in the vectors according to increasing node numbers. Because the JFET nodes are numbered sequentially, the corresponding transfer functions occupy successive elements in each vector. This method of storage allows the transfer functions to be accessed and modified by the following iterative procedure:

STEP 1: Initialization

a) Length of each T.F. vector: LT=2*ORDER

b) Location of NODE in first vector: I1=2*NODE-1

c) Location of NODE in last vector: $I2=I1+(2^{NORDR}-3)*LT$

d) Origin of scratch area on file 24: $L = 2^{NORDR} * LT$

e) Set index at first vector: I=I1

STEP 2: Access and Modify Next Vector

a) Read transfer functions at sequential device nodes

beginning at I, store in X, X1, and X2.

b) Write device voltages to scratch area at L, increment L=L+2.

c) Calculate X2=X-X2

d) Write updated transfer functions to file 24 beginning at I.

STEP 3: Increment and Test Index

a) Increment I=I+LT

b) If (I.LE.I2) more vectors remain to be modified. Go to Step 2.

c) Otherwise modified transfer function table complete.

The nonlinear currents of order NORDR for the JFET are defined as:

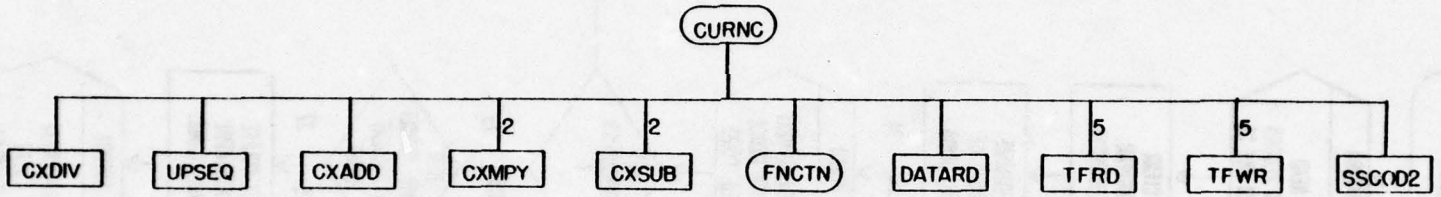
$$HG = \sum_{K=2}^{NORDR} A(K) * F_K^{NORDR} \quad (\overline{NODE + 3})$$

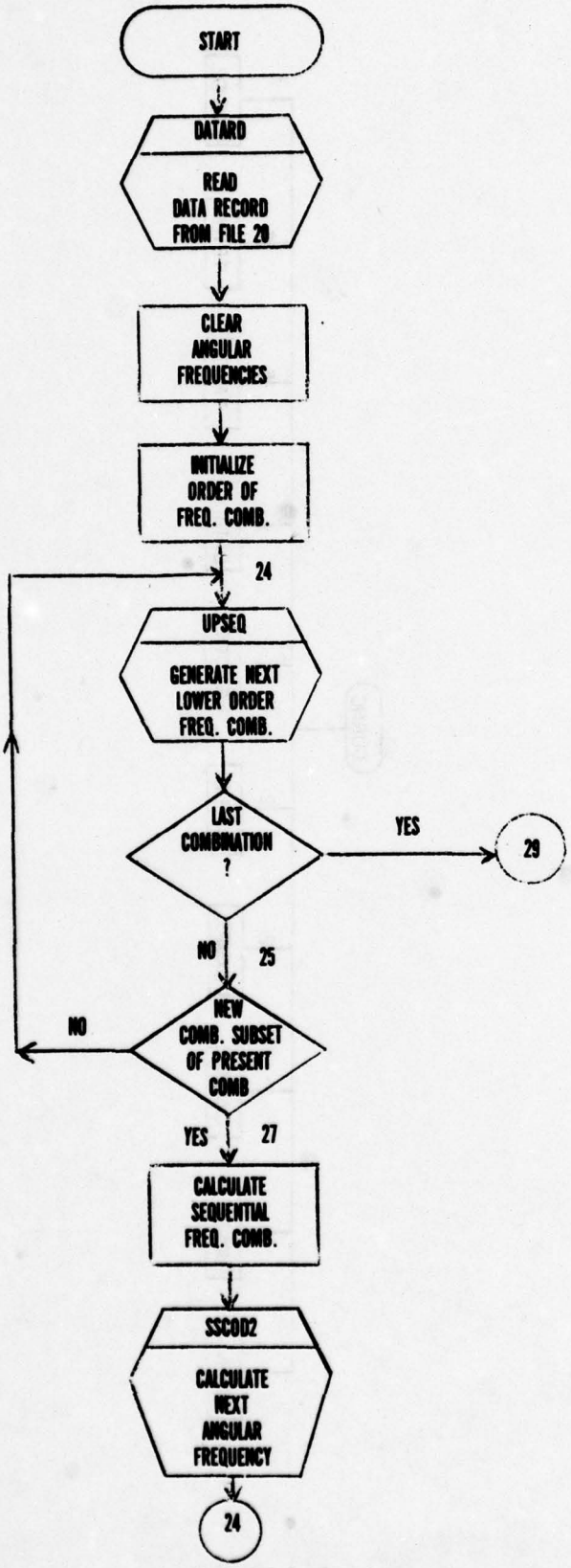
$$HL = \sum_{K=2}^{NORDR} S * C(K) * F_K^{NORDR} \quad (\overline{NODE + 3})$$

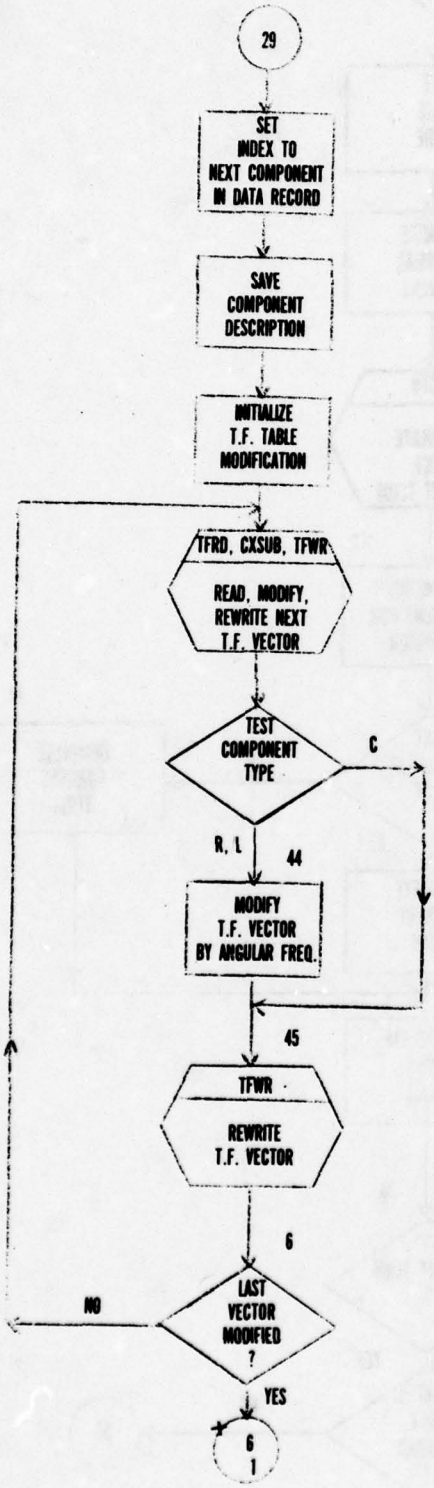
Subroutine CURJF initializes HG and HL to zero, then each symmetric transfer function expression $F_K^{NORDR}(NODE+3)$ for K=2,3,.. NORDR is computed by subroutine FNCTN, multiplied by the appropriate coefficients A(K) or S*C(K), and accumulated in HG and HL.

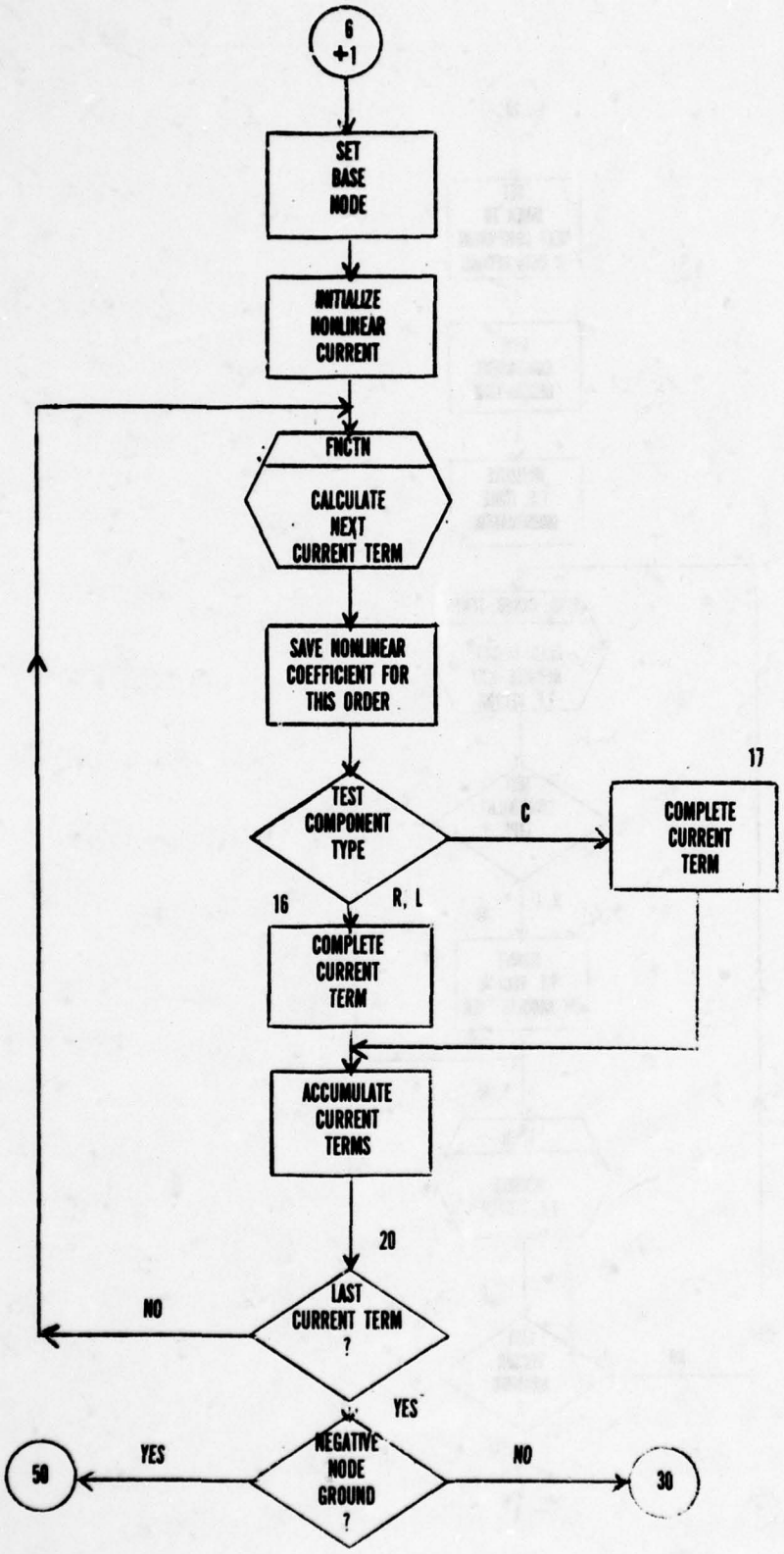
Then the transfer function table is restored by reading the transfer functions saved in the scratch area in file 24 and rewriting them to the original locations in the table. The procedure is similar to that employed for modifying the transfer function table.

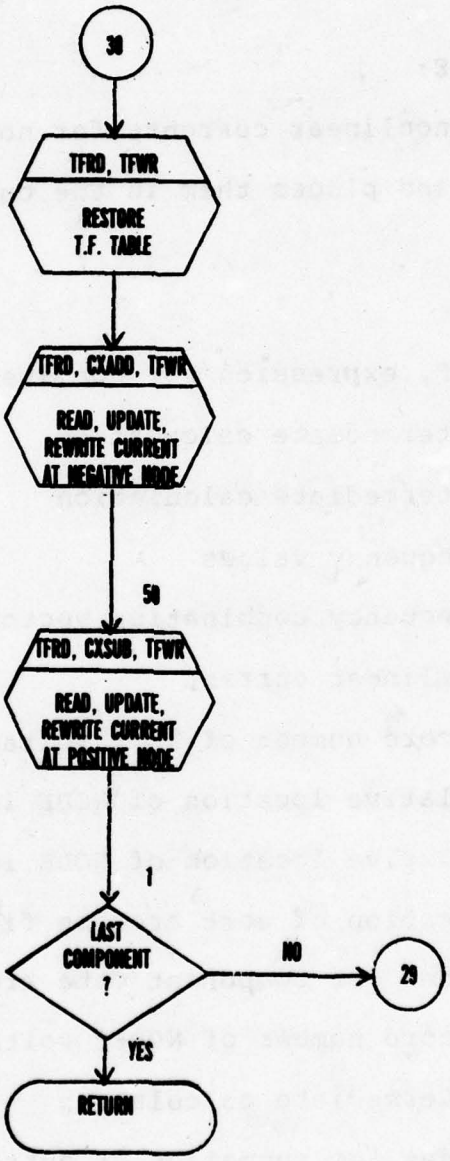
The nonlinear currents are placed in the current vector on file 25 by subtracting HL from the current at NODE, subtracting HG from the current at NODE+1, and adding HG and HL to the current at NODE+2. The current elements for the sequential JFET nodes, beginning at I=2*NODE-1, are read from file 25 by subroutine TFRD and stored in X, X1, and X2. HL is subtracted from X and HG is subtracted from X1 by subroutine CXSUB. HG and HL are added to X2 by subroutine CXADD. The updated current elements are rewritten to their original locations by subroutine TFWR and subroutine CURJF returns to the calling program.


CURNC
CXDIV
UPSEQ
CXADD
2
CXMPY
2
CXSUB
FNCTN
DATARD
5
TFRD
5
TFWR
SSCOD2


START
DATARD
READ DATA RECORD FROM FILE 20
CLEAR ANGULAR FREQUENCIES
INITIALIZE ORDER OF FREQ. COMB.
24
UPSEQ
GENERATE NEXT LOWER ORDER FREQ. COMB.
LAST COMBINATION ?
YES
29
NO
25
NEW COMB. SUBSET OF PRESENT COMB
NO
YES
27
CALCULATE SEQUENTIAL FREQ. COMB.
SSCOD2
CALCULATE NEXT ANGULAR FREQUENCY
24


29
SET INDEX TO NEXT COMPONENT IN DATA RECORD
SAVE COMPONENT DESCRIPTION
INITIALIZE T.F. TABLE MODIFICATION
TFRD, CXSUB, TFWR
READ, MODIFY, REWRITE NEXT T.F. VECTOR
TEST COMPONENT TYPE
C
R, L
44
MODIFY T.F. VECTOR BY ANGULAR FREQ.
45
TFWR
REWRITE T.F. VECTOR
6
LAST VECTOR MODIFIED ?
NO
YES
6
1


6
+1
SET
BASE
NODE
INITIALIZE
NONLINEAR
CURRENT
FNCTN
CALCULATE
NEXT
CURRENT TERM
SAVE NONLINEAR
COEFFICIENT FOR
THIS ORDER
17
TEST
COMPONENT
TYPE
C
COMPLETE
CURRENT
TERM
16
R, L
COMPLETE
CURRENT
TERM
ACCUMULATE
CURRENT
TERMS
20
LAST
CURRENT TERM
?
NO
YES
NEGATIVE
NODE
GROUND
?
YES
50
NO
30


30
TFRD, TFWR
RESTORE
T.F. TABLE
TFRD, CXADD, TFWR
READ, UPDATE,
REWRITE CURRENT
AT NEGATIVE MODE
50
TFRD, CXSUB, TFWR
READ, UPDATE,
REWRITE CURRENT
AT POSITIVE MODE
1
LAST
COMPONENT
?
NO
29
YES
RETURN

NAME: CURNC

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear currents for nonlinear components and places them in the current vector

VARIABLES:

A	= T.F. expression for nonlinear current
B	= Intermediate calculation
CONST	= Intermediate calculation
FREQ	= Frequency values
FRQ	= Frequency combination vector
HK	= Nonlinear current
I	= Record number of NODE voltage (current)
I1	= Relative location of NODE in first T.F.
I2	= Relative location of NODE in last T.F.
I3	= Location of work area on file 24
I4	= Index for component data area
J	= Record number of NODEB voltage
J2PI	= Intermediate calculation
K	= Index for summation of current calculation
L	= Record number for file 24 work area
LT	= Length of each T.F. vector in words
NCOMP	= Length of nonlinear component data record

NODE = Positive node of connection

NODEB = Negative node of connection

NORDR = Order of analysis

RCL = Component type; 1 = Resistor
2 = Capacitor
3 = Inductor

S = Angular frequency

SP = Frequency product

X = NODE voltage (current)

Y = NODEB voltage (current)

SUBROUTINES CALLED:

CXADD	CXDIV	CXMPY	CXSUB	DATARD
FNCTN	SSCOD2	TFRD	TFWR	UPSEQ

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURNC calculates the nonlinear currents for nonlinear components and places them in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear current to be calculated; FREQ, the input frequency values for the circuit; FRTBL, the frequency number table; SEQ, the frequency combination vector; and S, the angular frequency. The nonlinear current is a function of the device coefficients stored in the nonlinear components data

record, the node voltages derived from the lower order transfer functions stored on file 24, and, in the case of the nonlinear inductor, the angular frequency at the lower orders of analysis.

First the nonlinear components data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Then a table of lower order angular frequencies is created for each vector in the transfer function table. The method is to recreate every lower order frequency combination which is a subset of the present frequency combination vector SEQ, calculate the angular frequency for that combination, and store it in the vector FRTAB. The frequency combinations are recreated in the same order as the transfer function vectors were selected and stored in the transfer function table by subroutine CONTRL, so that the angular frequency stored at FRTAB(I) corresponds to the Ith transfer function vector on file 24.

The recreation of lower order frequency combinations is performed by repeated calls to subroutine UPSEQ under the control of the integer variable MORDR, which maintains the order of the combination being formed. Subroutine SSCOD2 calculates the angular frequency and appends it to the FRTAB vector. The complete procedure for creating the angular frequency table is summarized as follows·

STEP 1· Initialization

a) Initialize FRTAB subscript to LL=0.

b) Initialize maximum order of frequency combination required to MXORD=NORDR.

c) Initialize number of frequencies in lower order combination to NORDR=0.

STEP 2. Generate Next Frequency Combination

a) Call subroutine UPSEQ to generate next frequency combination vector in SEQ1.

b) If (MORDR.LT.NORDR), go to Step 3.

c) Otherwise angular frequency table complete.

STEP 3. Test for SEQ1 a Subset of SEQ

a) If every SEQ1(I) for I=1,2,...MORDR equals some SEQ(J) for J=1,2,...NORDR, go to Step 4.

b) Otherwise go to Step 2.

STEP 4. Calculate Angular Frequency and Append to Table

a) Calculate sequential frequency combination by:

$$SCOMB = \sum_{I=1}^{MORDR} SEQ1(I)$$

b) Increment FRTAB index to LL=LL+1.

c) Call subroutine SSCOD2 to calculate angular frequency and append to table at FRTAB(LL).

d) Go to Step 2.

For each nonlinear component description in the data record, subroutine CURNC transfers the component type to RCL and the positive and negative nodes of connection to NODE and NODEB respectively. Then the node voltages for the nonlinear current calculations are computed by replacing the transfer function at NODE with the difference between the transfer functions at NODE and NODEB for every vector in the transfer function table. For nonlinear inductors, the updated transfer function at NODE is divided by the corresponding angular frequency from the table FRTAB. This "modified transfer function table" serves as the basis of the nonlinear current calculations. If the negative node of connection is ground (NODEB=0) for any component in the data record, the modification of transfer functions for that component is by-passed.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the circuit and occupies two decimal storage locations. The transfer functions are stored in the vectors according to increasing node numbers. This method of storage allows the transfer functions to

be accessed and modified by the following iterative procedure:

STEP 1: Initialization

a) Length of each T.F. vector: LT=2*ORDER

b) Location of NODE in first vector: I1=2*NODE-1

c) Location of NODE in last vector: $I2=I1+(2^{NORDR}-3)*LT$

d) Origin of scratch area on file 24: $L=2^{NORDR}*LT$

e) Subscript for FRTAB vector: M=0

f) Set index at first vector: I=I1

STEP 2: Access and Modify Next Vector

a) Increment FRTAB subscript M=M+1

b) Read transfer function from I, store in X.

c) Write X to scratch area at L, increment L=L+2.

d) If NODEB=0, go to Step 2h.

e) Otherwise locate NODEB in vector: J=I+(NODEB-NODE)*2

f) Read transfer function from J, store in Y.

g) Calculate X=X-Y.

h) If component is inductor (RCL=3), calculate X=X/FRTAB(M)
Otherwise go to Step 3i.

i) Write X to file 24 at I.

STEP 3: Increment and Test Index

a) Increment I=I+LT for next vector

b) If (I.LE.I2), more vectors remain to be modified. Go to Step 2.

c) Otherwise modified transfer function table complete.

The nonlinear current of order NORDR for the nonlinear resistor (RCL=1) is defined as:

$$HK = \sum_{K=2}^{NORDR} CONST * F_K^{NORDR}(\overline{NODE})$$

For the nonlinear capacitor (RCL=2):

$$HK = \sum_{K=2}^{NORDR} CONST * S/K * F_K^{NORDR}(\overline{NODE})$$

For the nonlinear inductor (RCL=3):

$$HK = \sum_{K=2}^{NORDR} CONST * F_K^{NORDR}(\overline{NODE})$$

where CONST is the Kth nonlinear coefficient taken from the

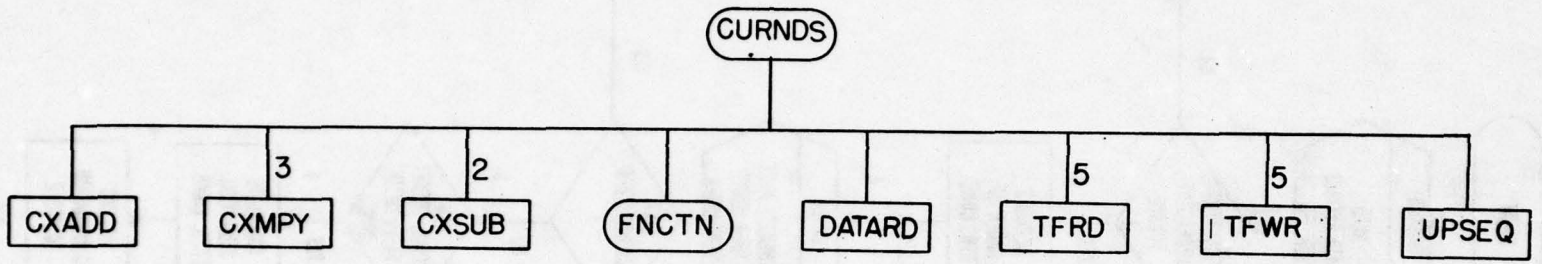
component description in BUFF.

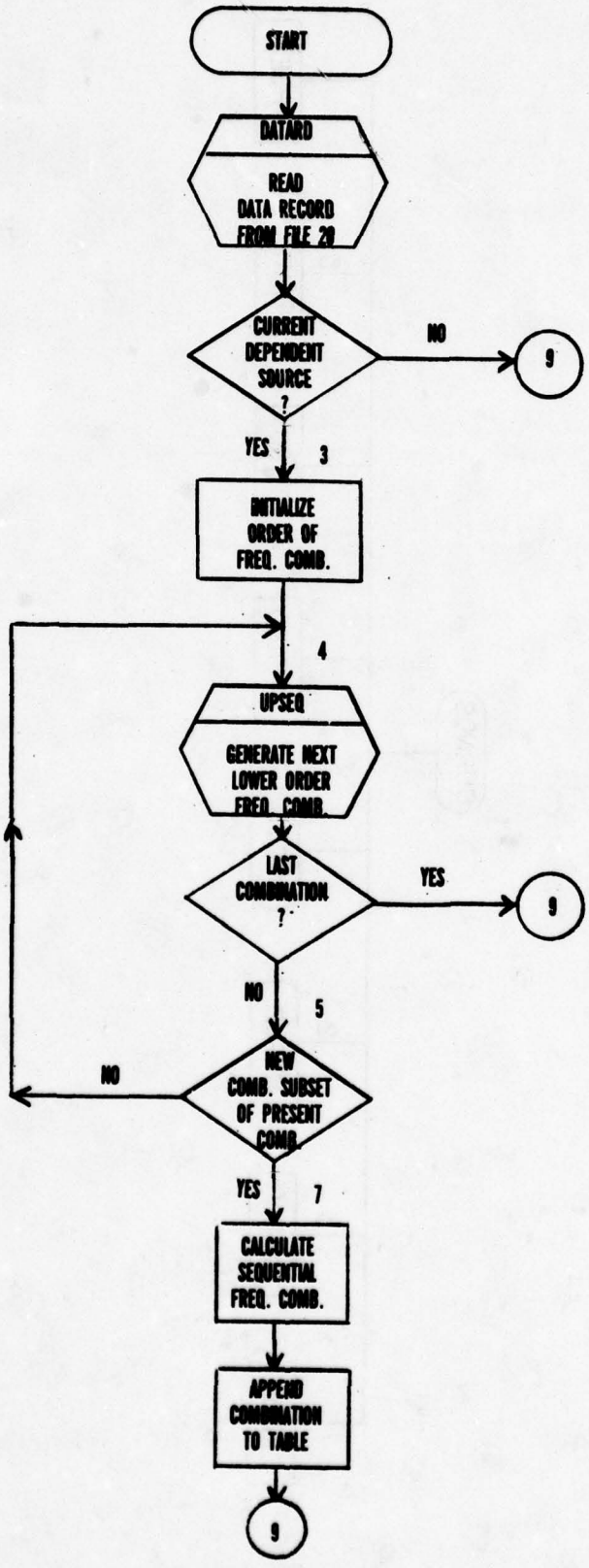
Subroutine CURNC initializes HK to zero, then each symmetric transfer function expression $F_K^{NORDR}(\overline{NODE})$ for K=2,3...NORDR is computed by subroutine FNCTN. After the intermediate calculation for CONST has been performed depending on the component type, the product is formed and accumulated in HK.

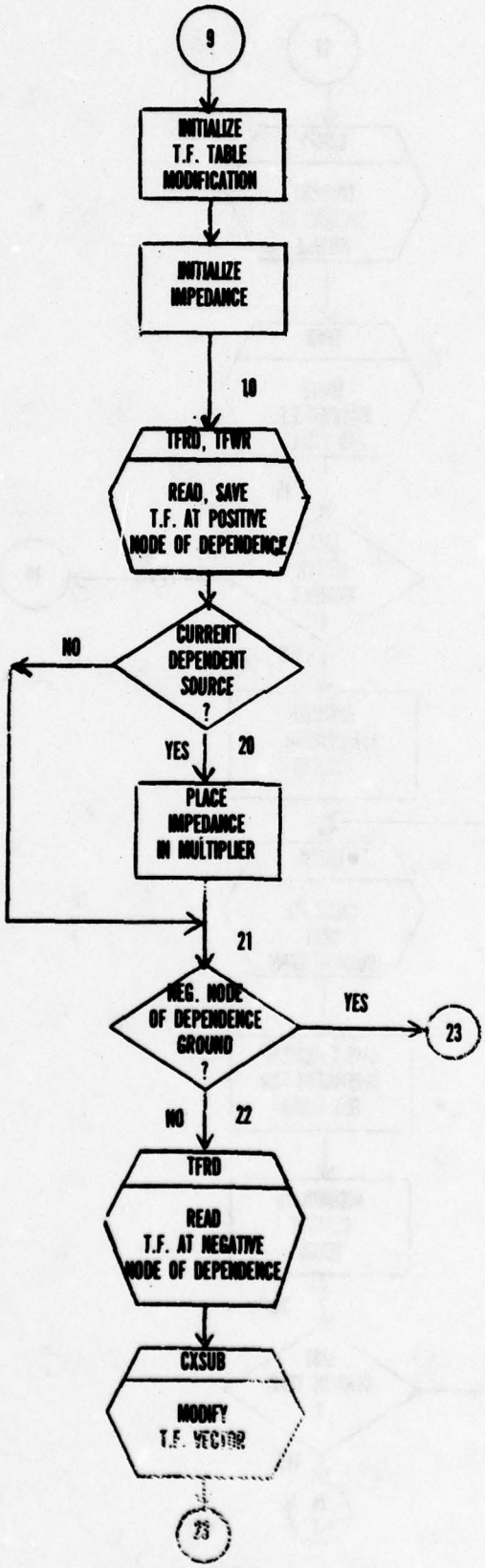
Then the transfer function table is restored by reading the transfer functions saved in the scratch area on file 24 and rewriting them to their original locations in the table. The procedure is similar to that employed for modifying the transfer function table. If the negative node of connection is ground (NODEP=Ø), the restore procedure is by-passed.

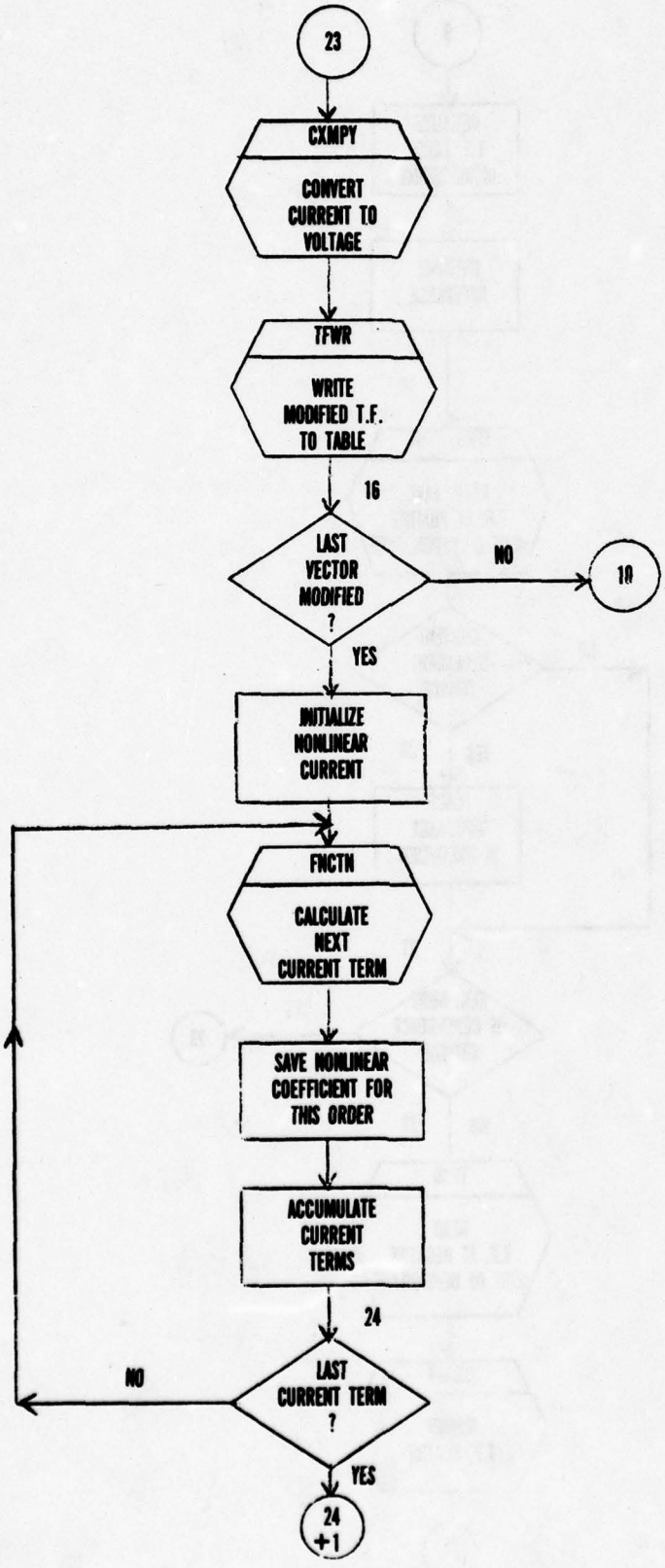
The nonlinear current is placed in the current vector on file 25 by adding it to the current at NODEB (for NODEB≠Ø) and subtracting it from the current at NODE. The NODEB current, located at I=2*NODEB-1, is read from file 25 by subroutine TFRD and stored in X. HK is added to X by subroutine CXADD and the updated NODEB current is rewritten to its original location by subroutine TFWR. The procedure is repeated for the NODE current, except that the NODE current is located at I=2*NODE-1 and HK is subtracted from X by subroutine CXSUB.

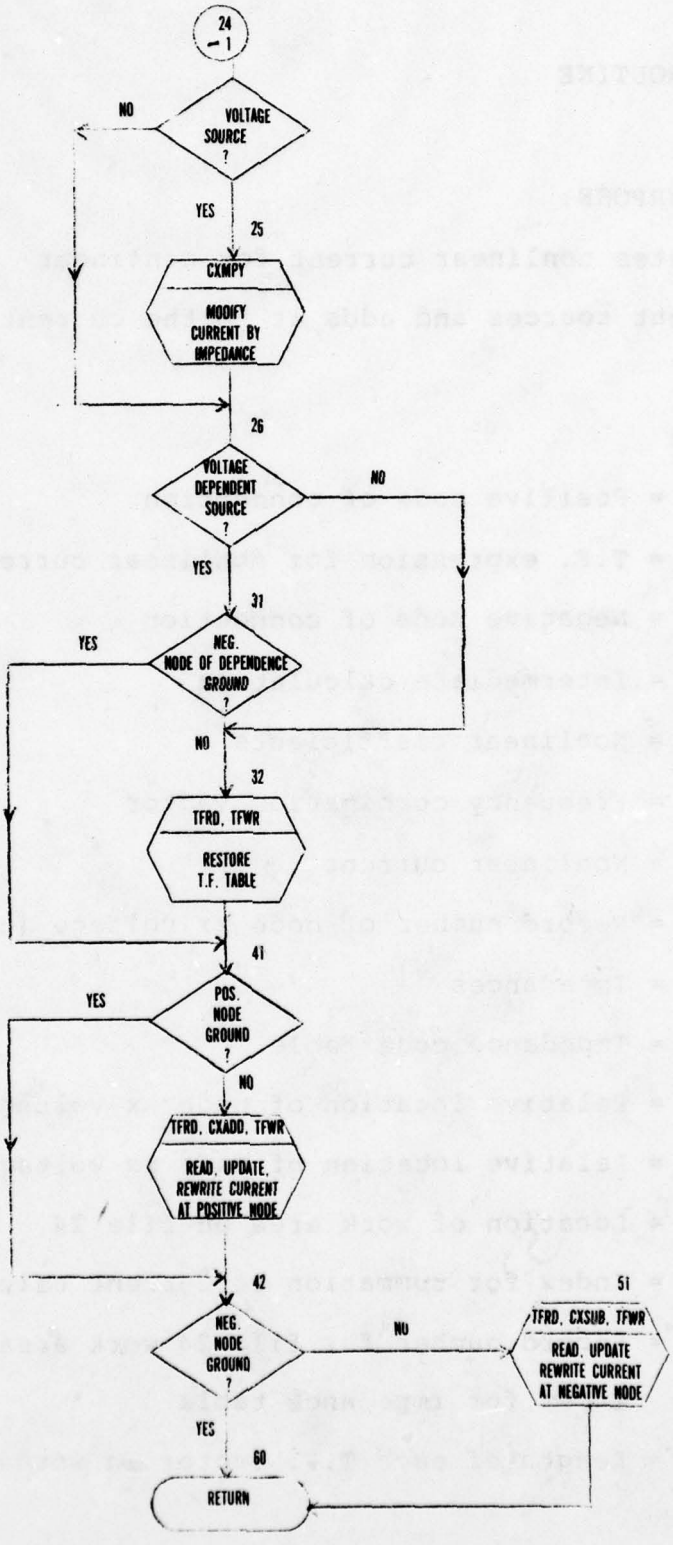
After the nonlinear currents have been calculated and placed in the current vector for every component in the data record, subroutine CURNC returns to the calling program.


CURNDS
CXADD
3
CXMPY
2
CXSUB
FNCTN
DATARD
5
TFRD
5
TFWR
UPSEQ


START
DATARD
READ DATA RECORD FROM FILE 20
CURRENT DEPENDENT SOURCE ?
NO
9
YES
3
INITIALIZE ORDER OF FREQ. COMB.
4
UPSEQ
GENERATE NEXT LOWER ORDER FREQ. COMB.
LAST COMBINATION ?
YES
9
NO
5
NEW COMB. SUBSET OF PRESENT COMB.
NO
YES
7
CALCULATE SEQUENTIAL FREQ. COMB.
APPEND COMBINATION TO TABLE
9


9
INITIALIZE T.F. TABLE MODIFICATION
INITIALIZE IMPEDANCE
10
TFRD, TFWR
READ, SAVE T.F. AT POSITIVE NODE OF DEPENDENCE
CURRENT DEPENDENT SOURCE ?
NO
YES
20
PLACE IMPEDANCE IN MULTIPLIER
21
NEG. NODE OF DEPENDENCE GROUND ?
YES
23
NO
22
TFRD
READ T.F. AT NEGATIVE NODE OF DEPENDENCE
CXSUB
MODIFY T.F. VECTOR
23


23
CXMPY
CONVERT CURRENT TO VOLTAGE
TFWR
WRITE MODIFIED T.F. TO TABLE
16
LAST VECTOR MODIFIED ?
NO
18
YES
INITIALIZE NONLINEAR CURRENT
FNCTN
CALCULATE NEXT CURRENT TERM
SAVE NONLINEAR COEFFICIENT FOR THIS ORDER
ACCUMULATE CURRENT TERMS
24
LAST CURRENT TERM ?
NO
YES
24 +1


24
-1
VOLTAGE SOURCE ?
NO
YES
25
CXMPY
MODIFY CURRENT BY IMPEDANCE
26
VOLTAGE DEPENDENT SOURCE ?
NO
YES
31
NEG. NODE OF DEPENDENCE GROUND ?
YES
NO
32
TFRD, TFWR
RESTORE T.F. TABLE
41
POS. NODE GROUND ?
YES
NO
TFRD, CXADD, TFWR
READ, UPDATE, REWRITE CURRENT AT POSITIVE NODE
42
NEG. NODE GROUND ?
NO
51
TFRD, CXSUB, TFWR
READ, UPDATE REWRITE CURRENT AT NEGATIVE NODE
YES
60
RETURN

NAME: CURNDS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear current for nonlinear dependent sources and adds it to the current vector

VARIABLES:

A	= Positive node of connection
AA	= T.F. expression for nonlinear current
B	= Negative node of connection
CC	= Intermediate calculation
COEF	= Nonlinear coefficients
FREQ	= Frequency combination vector
HK	= Nonlinear current
I	= Record number of node xx voltage (current)
IMPS	= Impedances
IMTBL	= Impedance code table
I1	= Relative location of node xx voltage in first T.F.
I2	= Relative location of node xx voltage in last T.F.
I3	= Location of work area on file 24
K	= Index for summation of current calculation
L	= Record number for file 24 work area and index for impedance table
LT	= Length of each T.F. vector in words

M = Index for impedance code table

MORDR = Order of impedance

MXORD = Maximum order of analysis

NORDR = Order of analysis

S = Angular frequency

SCOMP = Sequential frequency combination

SEQ = Frequency combination vector

SEQ1 = Lower-order frequency combination vector

SOURCE = Type of source: 1 = VC

2 = CC

3 = VV

4 = CV

X = Node XX voltage (current)

XX = Positive node of dependence

Y = Node YY voltage (current)

YY = Negative node of dependence

Z = Impedance value

SUBROUTINES CALLED:

CXADD CXMPY CXSUB DATARD FNCTN

TFRD TFWR UPSEQ

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURNDS calculates the nonlinear currents for

nonlinear dependent sources and places them in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear current to be calculated; S, the angular frequency; and SEQ, the frequency combination vector. The nonlinear current is a function of the device coefficients stored in the nonlinear dependent source data record and the node voltages derived from the lower order transfer functions stored on file 24. For current dependent sources, the nonlinear current also depends on the impedance at the nodes of dependence, while for voltage sources, the nonlinear current depends on the impedance at the nodes of connection. The positive and negative nodes of dependence (XX and YY), the positive and negative nodes of connection (A and B), and the type of SOURCE are stored in the nonlinear dependent source driver record in global common.

First the nonlinear dependent source data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Then for current dependent sources only (SOURCE=2 or 4), a table of lower order sequential frequency combinations is created for each vector in the transfer function table. The method is to recreate every lower order frequency combination which is a subset of the frequency combination vector SEQ, calculate the sequential frequency combination, and store it in the vector IMTBL. The frequency combinations are recreated in the same order as the transfer function vectors were selected and stored in the transfer function table by subroutine CONTRL, so that the sequential frequency combination IMTBL(I) corresponds to the Ith transfer function vector on file 24.

The recreation of lower order frequency combinations is performed by repeated calls to subroutine UPSEQ under the control of the integer variable MORDR, which maintains the order of the combination being formed. The complete procedure for creating the sequential frequency combination table is summarized as follows:

STEP 1· Initialization

a) Initialize INTEL index to L=Ø.

b) Initialize maximum order of frequency combination required to MXORD=NORDR.

c) Initialize number of frequencies in lower order combination to MORDR=Ø.

STEP 2· Generate Next Frequency Combination

a) Call subroutine UPSEQ to generate next frequency combination vector in SEQ1.

b) If (MORDR.LT.NORDR), go to Step 3.

c) Otherwise sequential frequency combination table complete.

STEP 3· Test for SEQ1 a Subset of SEQ

a) If every SEQ1(I) for I=1,2,...MORDR equals some SEQ(J) for J=1,2,...NORDR, go to Step 4.

b) Otherwise go to Step 2.

STEP 4: Calculate Sequential Frequency Combination and Append to Table

a) Calculate sequential frequency combination by:

$$SCOMB = \sum_{I=1}^{MORDR} SEQ1(I)$$

b) Increment IMTBL index to L=L+1.

c) Append sequential frequency combination to table by IMTBL(L)=SCOMB.

d) Go to Step 2.

Then the node voltages for the nonlinear current calculations are computed by replacing the transfer function at node XX with the difference between the transfer functions at nodes XX and YY for every vector in the transfer function table. For current dependent sources (SOURCE=2 or 4), the updated transfer function vector at node XX is divided by the corresponding impedance IMPS(IMTBL(M)) from the nonlinear

dependent source data record. This "modified transfer function table" serves as the basis for the nonlinear current calculations. If the negative node of dependence is ground (YY=Ø), the subtraction of transfer functions is by-passed.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the circuit and occupies two decimal storage locations. The transfer functions are stored in the vectors according to increasing node numbers. This method of storage allows the transfer functions to be accessed and modified by the following iterative procedure:

STEP 1: Initialization

a) Subscript for IMTBL vector: M=Ø.

b) Location of node XX in first vector: I1=2*XX-1.

c) Length of each T.F. vector: LT=2*ORDER

d) Location of node XX in last vector
I2=I1+(2^{NORDR}-3)*LT.

e) Origin of scratch area on file 24: $L=2^{NORDR}$ *LT.

f) Impedance value: Z=(1., Ø.)

g) Set index at first vector: I=I1.

STEP 2: Access and Modify Next Vector

a) Read transfer function from I, store in X.

b) Increment IMTBL subscript M=M+1

c) Write X to scratch area at L, increment L=L+2.

d) For current dependent sources (SOURCE=2 or 4), store impedance value Z=IMPS(IMTBL(M)).

e) If node YY=0, go to Step 2i.

f) Otherwise locate node YY in vector: J=I+(YY-XX)*2.

g) Read transfer function from J, store in Y.

h) Calculate X=X-Y

i) Calculate X=X/Z

j) Write X to file 24 at I.

STEP 3: Increment and Test Index

a) Increment I=I+LT

b) If (I.LE.I2), more vectors remain to be modified. Go to Step 2.

c) Otherwise modified transfer function table complete.

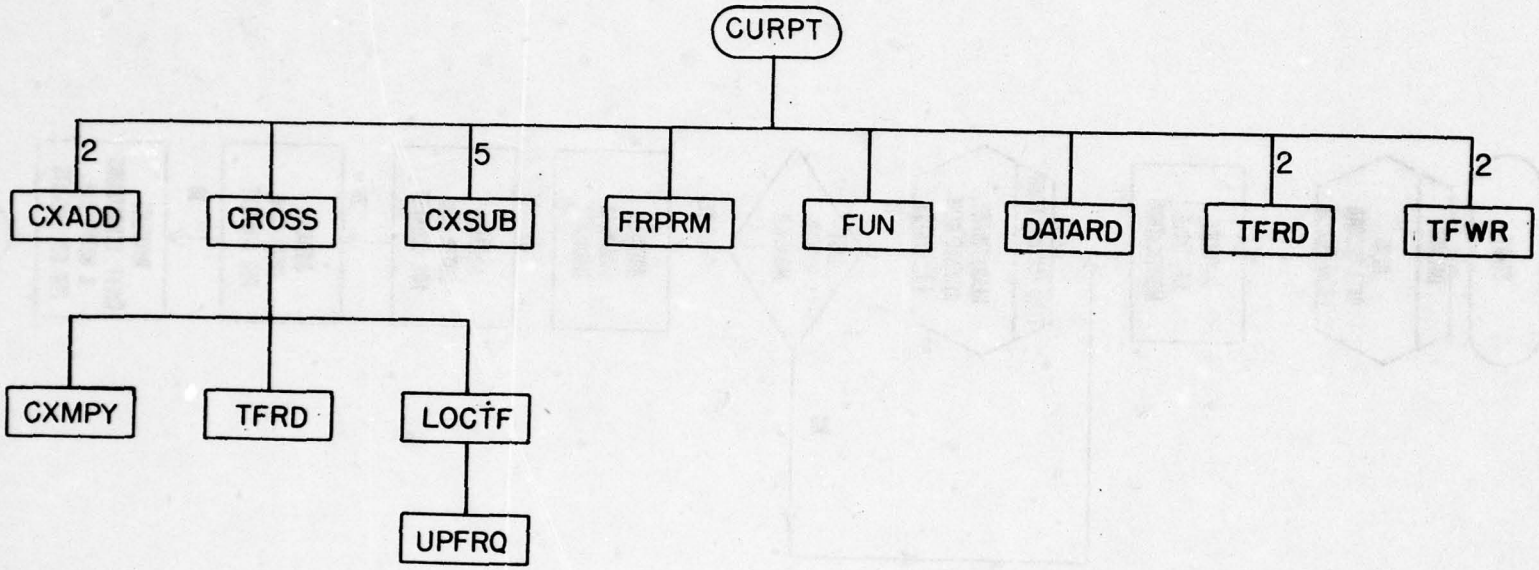
The nonlinear current of order NORDR for the nonlinear dependent sources is defined as:

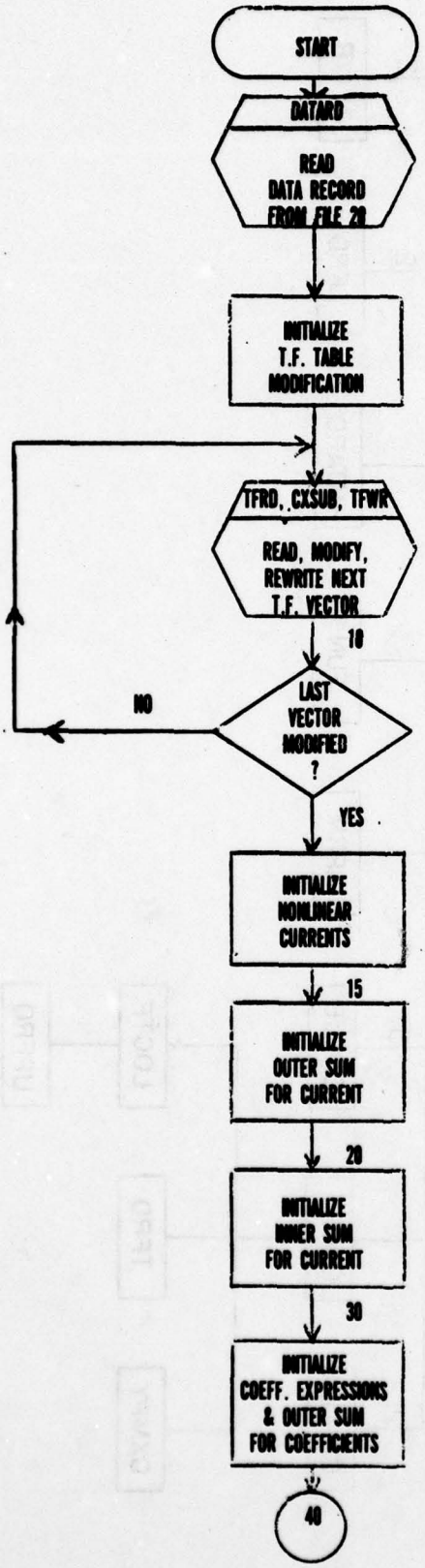
$$HK = \sum_{K=2}^{NORDR} COEF(K) * F_K^{NORDR}(\overline{XX})$$

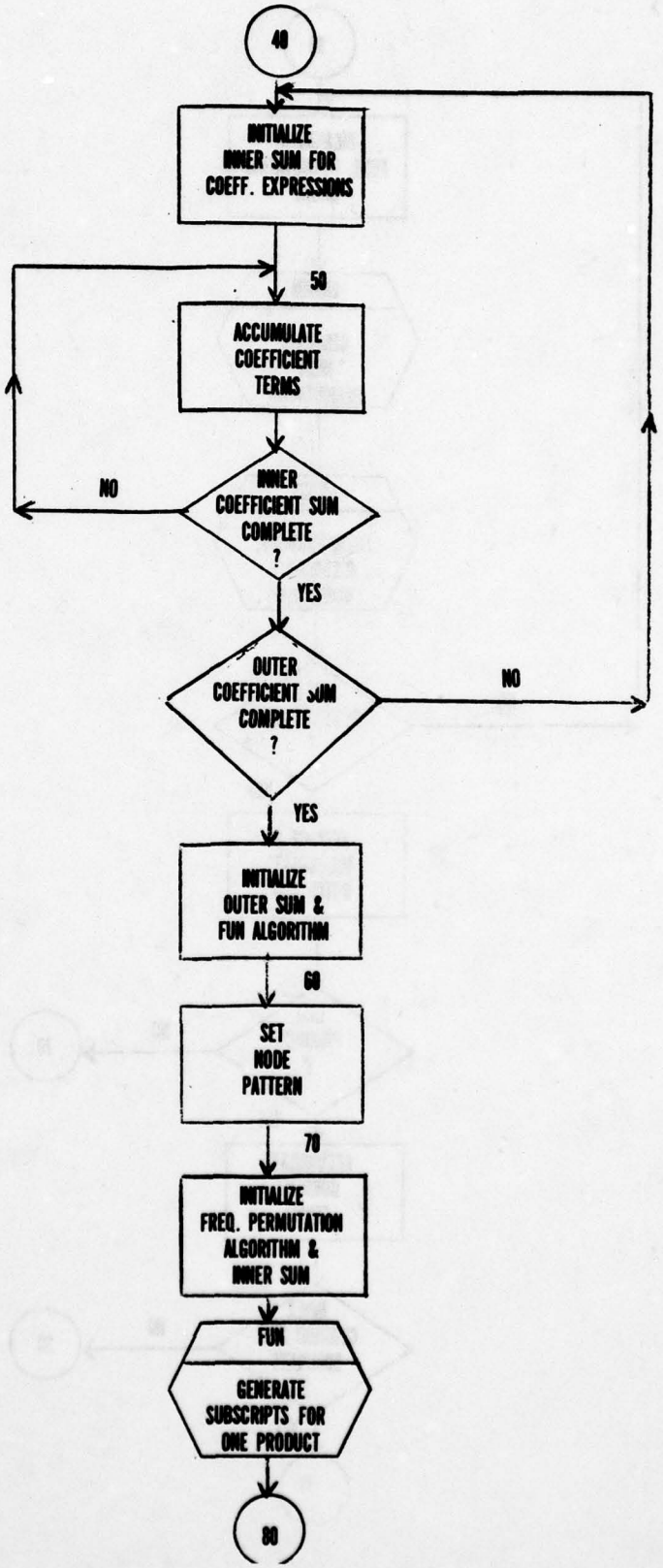
Subroutine CURNDS initializes HK to zero, then each symmetric transfer function expression $F_K^{NORDR}(\overline{XX})$ for K=2,3,...NORDR is computed by subroutine FNCTN, multiplied by the appropriate coefficient COEF(K), and accumulated in HK. For voltage sources only (SOURCE=3 or 4), HK is divided by the impedance at the nodes of connection, Z = (1.ØE1Ø., Ø.).

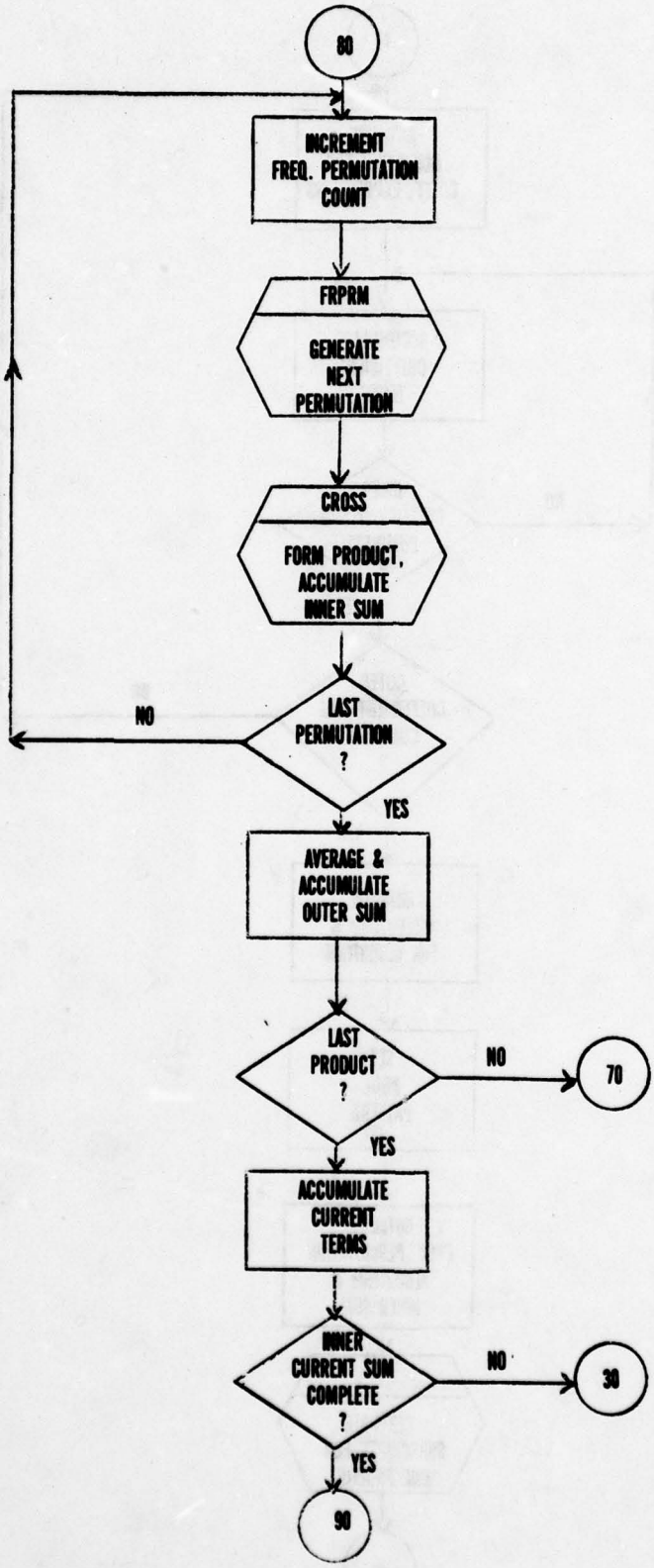
Then the transfer function table is restored by reading the transfer functions saved in the scratch area on file 24 and rewriting them to their original locations in the table. The procedure is similar to that employed for modifying the transfer function table. If the negative node of dependence is ground (YY=Ø), the restore procedure is by-passed.

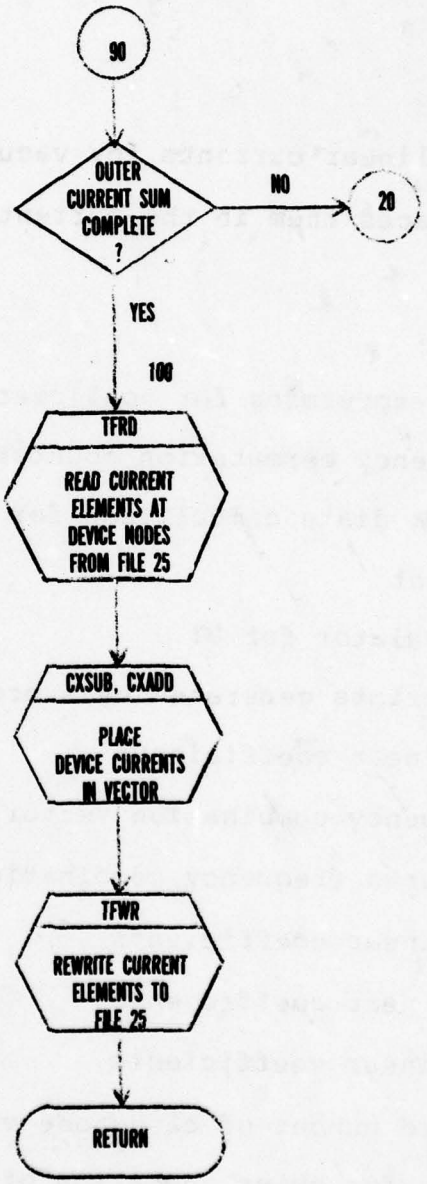
The nonlinear current is placed in the current vector on file 25 by adding it to the current at the positive node of connection A (for A≠Ø) and subtracting it from the negative node of connection B (for B≠Ø). The node A current, located at I=2*A-1, is read from file 25 by subroutine TFRD and stored in X. HK is added to X by subroutine CXADD and the updated node A current is rewritten to its original location by subroutine TFWR. The procedure is repeated for the node B current, except that node B is located at I=2*B-1 and HK is subtracted from X by subroutine CXSUB. Subroutine CURNDS then returns to the calling program.


CURPT
2
CXADD
CROSS
5
CXSUB
FRPRM
FUN
DATARD
2
TFRD
2
TFWR
CXMPY
TFRD
LOCTF
UPFRQ


START
DATARD
READ
DATA RECORD
FROM FILE 20
INITIALIZE
T.F. TABLE
MODIFICATION
TFRD, CXSUB, TFWR
READ, MODIFY,
REWRITE NEXT
T.F. VECTOR
10
LAST
VECTOR
MODIFIED
?
NO
YES
INITIALIZE
NONLINEAR
CURRENTS
15
INITIALIZE
OUTER SUM
FOR CURRENT
20
INITIALIZE
INNER SUM
FOR CURRENT
30
INITIALIZE
COEFF. EXPRESSIONS
& OUTER SUM
FOR COEFFICIENTS
40

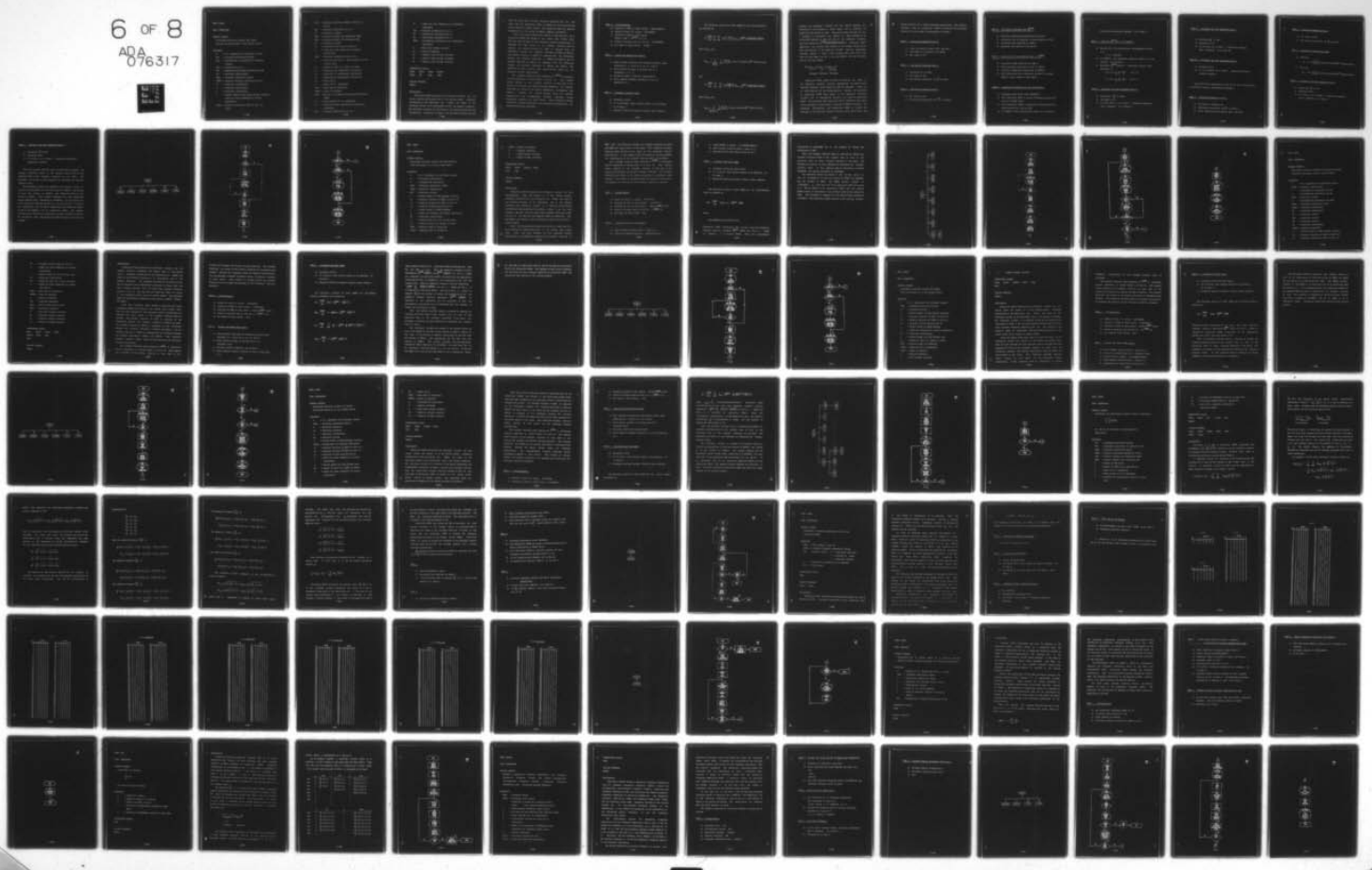

40
INITIALIZE
INNER SUM FOR
COEFF. EXPRESSIONS
50
ACCUMULATE
COEFFICIENT
TERMS
INNER
COEFFICIENT SUM
COMPLETE
?
NO
YES
OUTER
COEFFICIENT SUM
COMPLETE
?
NO
YES
INITIALIZE
OUTER SUM &
FUN ALGORITHM
60
SET
NODE
PATTERN
70
INITIALIZE
FREQ. PERMUTATION
ALGORITHM &
INNER SUM
FUN
GENERATE
SUBSCRIPTS FOR
ONE PRODUCT
80


80
INCREMENT FREQ. PERMUTATION COUNT
FRPRM
GENERATE NEXT PERMUTATION
CROSS
FORM PRODUCT, ACCUMULATE INNER SUM
LAST PERMUTATION ?
NO
YES
AVERAGE & ACCUMULATE OUTER SUM
LAST PRODUCT ?
NO
70
YES
ACCUMULATE CURRENT TERMS
INNER CURRENT SUM COMPLETE ?
NO
30
YES
90


90
OUTER CURRENT SUM COMPLETE ?
NO
20
YES
100
TFRD
READ CURRENT ELEMENTS AT DEVICE NODES FROM FILE 25
CXSUB, CXADD
PLACE DEVICE CURRENTS IN VECTOR
TFWR
REWRITE CURRENT ELEMENTS TO FILE 25
RETURN

AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

6 OF 8
ADA 076317



NAME: CURPT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear currents for vacuum pentode and places them in the current vector

VARIABLES:

A = T.F. expression for nonlinear current

CNT = Frequency permutation counter

COEF = Intermediate calculation for nonlinear current

FACT = Accumulator for N!

FNK = Subscripts generated by subroutine FUN

FP = Nonlinear coefficients

FREQ = Frequency combination vector

FREQC = Permuted frequency combination vector

F2 = Nonlinear coefficients

G = Nonlinear coefficients

H = Nonlinear coefficients

I = Record number of base node voltage (current); index for outer summation of current calculation

IHIGH = Relative location of NODE in last T.F. vector

ILCW = Relative location of NODE in first T.F. vector

INC = Binomial coefficient for I,N

IP = Nonlinear current

ISWF = Control variable for subroutine FRPRM

ISWT = Control variable for subroutine FUN

I2 = Nonlinear current

I3 = Location of work area on file 24

J = Index for inner summation of current calculation

JNIC = Trinomial coefficient for J, N-I

L = Index for QQ vector. Record number for file 24 work area.

LT = Length of each T.F. vector in words

L1 = Subscript for coefficient calculation

L2 = Subscript for coefficient calculation

L3 = Subscript for coefficient calculation

MU = Input value

N = Order of T.F. expression

NODE = Base node of connection

NORDR = Order of analysis

PROD = Sum of T.F. products returned by subroutine CROSS

QQ = Node pattern of T.F. expression

R = Index for inner summation of coefficient expression

RJC = Binomial coefficient for R,J

S = Index for outer summation of coefficient expression

SIC = Binomial coefficient for S,I

SP = Coefficient expression for IP

S2 = Coefficient expression for I2

TERM = Intermediate calculation for coefficient expression

X = Base node voltage (current)

XX = Angular frequency

X1 = (Base+1) node voltage (current)

X2 = (Base+2) node voltage (current)

X3 = (Base+3) node voltage (current)

SUBROUTINES CALLED:

CROSS	CXADD	CXSUB	DATARD
FRPRM	FUN	TFRD	TFWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURPT calculates the nonlinear currents for the vacuum pentode and places them in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear current to be calculated and XX, the angular frequency. The nonlinear currents are functions of the device parameters and coefficients calculated in Phase 1 and the node voltages derived

from the lower order transfer functions stored on file 24. The base node of connection, NODE, is stored in the vacuum pentode driver record in global common. The remaining nodes are numbered sequentially by the system as NODE+1, NODE+2, and NODE+3.

First the vacuum pentode data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Then the node voltages for the nonlinear current calculations are computed for each vector in the transfer function table as follows: the transfer function at NODE is replaced with the difference between the transfer functions at NODE and NODE+3; the transfer function at NODE+1 is replaced with the difference between the transfer functions at NODE+1 and NODE+3; and the transfer function at NODE+2 is replaced with the difference between the transfer functions at NODE+2 and NODE+3. This "modified transfer function table" serves as the basis for the nonlinear current calculations.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the circuit and occupies two decimal storage locations. The transfer functions are stored in the vector according to increasing node numbers. Because the vacuum pentode nodes are numbered sequentially, the corresponding transfer functions occupy successive elements in each vector. This method of storage allows the transfer functions to be accessed and modified by the following iterative procedure.

STEP 1: Initialization

a) Location of NODE in first vector: ILOW=2*NODE-1.

b) Length of each T.F. vector: LT=2*ORDER.

c) Location of NODE in last vector:

IHIGH = ILOW + (2^{NORDR} -3) * LT

d) Origin of scratch area on file 24: I3=2^{NORDR}*LT

e) Set index to first vector: I=ILOW.

STEP 2: Access and Modify Next Vector

a) Read transfer functions for sequential device nodes beginning at I, store in X, X1, X2, and X3.

b) Write device voltages to scratch area at L, increment L = L + 2.

c) Calculate X=X-X3, X1=X1-X3, and X2=X2-X3.

d) Write updated transfer functions to file 24 beginning at I.

STEP 3: Increment and Test Index

a) Increment I=I+LT.

b) If (I.LE.IHIGH), more vectors remain to be modified. Go to Step 2.

c) Otherwise modified transfer function table complete.

The nonlinear currents of order NORDR for the vacuum pentode are defined as:

$$I2 = \sum_{N=2}^{NORDR} \sum_{I=0}^{N} \sum_{J=0}^{N-I} \frac{1}{N!} \binom{N}{I} \binom{N-I}{J} S2_{N,I,J} F_N^{NORDR} \overline{Q[NODE, NODE+1, NODE+2]}$$

where $S2_{N,I,J}$ is:

$$S2_{N,I,J} = \sum_{\substack{S=MAX \\ (I=1,0)}}^{I} \sum_{R=0}^{J} \binom{I}{S} \binom{J}{R} G(I-S+1)\, H(J-R+S+1)\, MU^{R-J}\, F2(N-I-J+1, R+1)$$

and

$$IP = \sum_{N=2}^{NORDR} \sum_{I=0}^{N} \sum_{J=0}^{N-I} \frac{1}{N!} \binom{N}{I} \binom{N-I}{J} SP_{N,I,J} F_N^{NORDR} \overline{Q[NODE, NODE+1, NODE+2]}$$

where $SP_{N,I,J}$ is:

$$SP_{N,I,J} = \sum_{\substack{S= \\ MAX(I-1,0)}}^{I} \sum_{R=0}^{J} \binom{I}{S} \binom{J}{R} G(I-S+1)\, H(J-R+S+1) MU^{R-J}\, FP(N-I-J+1, R+1)$$

Because the nonlinear currents for the vacuum pentode are functions of the voltages at three nodes, the Q function of subroutine QFN cannot be used. Subroutine CURPT contains its own Q function to determine the pattern of node voltages to be multiplied together to form one term of the symmetric transfer function expression $F_N^{NORDR} \overline{Q(NODE,NODE+1,NODE+2)}$. Subroutine CURVT apportions the relative node numbers to the integer vector QQ so that each product in the symmetric transfer function expression contains exactly N lower order voltages where the first I are from the base NODE, the next J are from NODE+2, and the remaining N-(I+J) are from NODE+1.

$$QQ=(\underbrace{1,1,\ldots 1}_{I},\underbrace{3,3,\ldots 3}_{J},\underbrace{2,2,\ldots 2}_{N-(I+J)})$$

Voltages Voltages Voltages

Subroutine FNCTN, which is used to calculate the terms of the symmetric transfer function expressions for NCAP's other nonlinear devices, calls subroutine QFN and therefore cannot be be used directly for the vacuum pentode. However, the algorithm of subroutine FNCTN does apply to the pentode's nonlinear currents and is implemented directly as a part of subroutine CURPT. The algorithm uses subroutine FUN to determine the orders of the voltages in each product, subroutine FRPRM to permute the frequency combination, and subroutine CROSS to access the voltages in the modified transfer function table and to form the

voltage product for a single frequency permutation. The complete algorithm used by subroutine CURPT to calculate the nonlinear currents of order NORDR is summarized as follows:

STEP 1: Initialize Summation Over N

a) Clear nonlinear currents I2=0. and IP=0.

b) Initialize N! accumulator to FACT=1

c) Set index at N=2

STEP 2: Initialize Summation Over I

a) Accumulate N! in FACT

b) Set index at I=0

c) Initialize accumulator for $\binom{N}{I}$ to INC=1

STEP 3: Initialize Summation Over J

a) Set index at J=0

b) Initialize accumulator for $\binom{N-I}{J}$ to JNIC=1

STEP 4: Initialize Algorithm for F_N^{NORDR}

a) Calculate coefficient expressions S2 and SP

b) Initialize sum of symmetric voltage products to A=Ø.

c) Initialize FUN algorithm by ISWT=-1

d) Establish node pattern in QQ vector

STEP 5: Initialize for one Additive Term of F_N^{NORDR}

a) Initialize FRPRM algorithm by ISWF=-1

b) Clear frequency permutation count to CNT=Ø.

c) Initialize symmetric accumulator to PROD=Ø.

d) Call subroutine FUN to determine pattern of voltage orders, return orders in FNK vector.

STEP 6: Accumulate Products over all Permutations

a) Increment permutation count CNT=CNT+1.

b) Call subroutine FRPRM to create frequency permutation, return in FREQC vector

c) Call subroutine CROSS to calculate voltage product for given permutation, add to PROD

d) If (ISWF=Ø), more permutations remain; go to Step 6a.

Otherwise permutations complete. Go to Step 7.

STEP 7: Test for F_N^{NORDR} at I, J Complete

a) Average over all permutations, add symmetric product to A:

$$A = A + PROD/CNT$$

b) If (ISWT=Ø), more symmetric products remain to be calculated. Go to Step 5.

c) Otherwise F_N^{NORDR} complete. Accumulate current terms according to:

$$I2 = I2 + \frac{1}{N!} \binom{N}{I} \binom{N-I}{J} * S2 * A$$

$$IP = IP + \frac{1}{N!} \binom{N}{I} \binom{N-I}{J} * SP * A$$

STEP 8: Increment and Test Summation Over J

a) Accumulate $\binom{N-I}{J}$ in JNIC

b) Increment J=J+1

c) If (J.LE.(N-I)), go to Step 4. Otherwise summation over J complete. Go to Step 9.

STEP 9: Increment and Test Summation over I

a) Accumulate $\binom{N}{I}$ in INC

b) Increment I=I+1

c) IF (I.LE.N), go to Step 3. Otherwise summation over I complete. Go to Step 10.

STEP 10: Increment and Test Summation over N

a) Increment N=N+1

b) If (N.LE.NORDR), go to STEP 2. Otherwise nonlinear currents complete.

The coefficient expressions S2 and SP are also calculated by an iterative process, summarized as follows:

STEP 1: Initialize Summation over S

a) Set index at S=MAX(I-1,0)

b) Initialize accumulator for $\binom{I}{S}$ to SIC=S+1

c) Clear coefficient expressions S2=0. and SP=0.

STEP 2: Initialize summation over R

a) Set index at R=0

b) Initialize accumulator for $\binom{J}{R}$ to RJC=1

STEP 3: Accumulate Coefficient Terms

a) Calculate

$$S2 = S2 + \binom{I}{S}\binom{J}{R}\ G(I-S+1)\ H(J-R+S+1)MU^{R-J}\ F2(N-I-J+1,R+1)$$

b) Calculate

$$SP = SP + \binom{I}{S}\binom{J}{R}\ G(I-S+1)\ H(J-R+S+1)\ MU^{R-J}\ FP(N-I-J+1,R+1)$$

STEP 4: Increment and Test Summation over R

a) Accumulate $\binom{J}{R}$ in RJC

b) Increment R=R+1

c) If (R.LE.J), go to Step 3. Otherwise summation over R complete, go to Step 5.

STEP 5: Increment and Test Summation over S

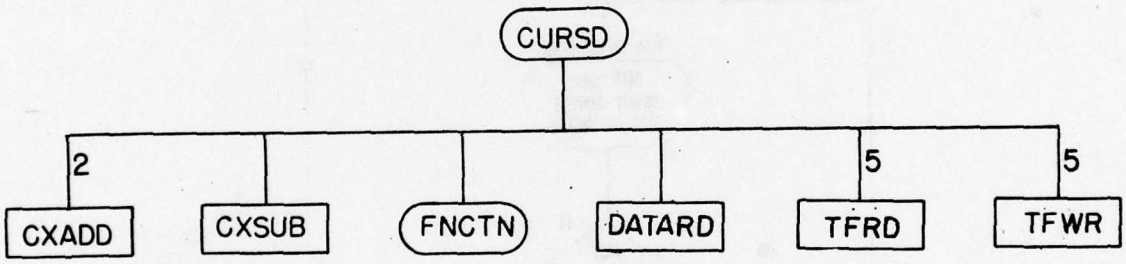
a) Accumulate $\binom{I}{S}$ in SIC

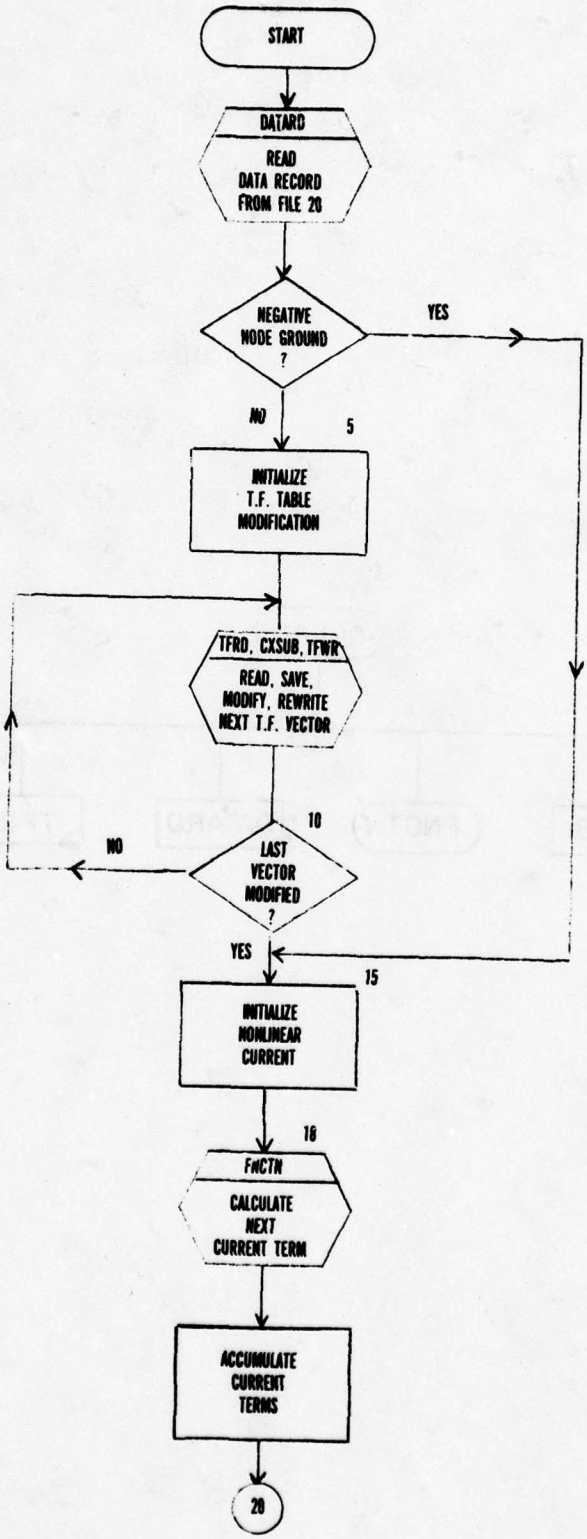
b) Increment S=S+1

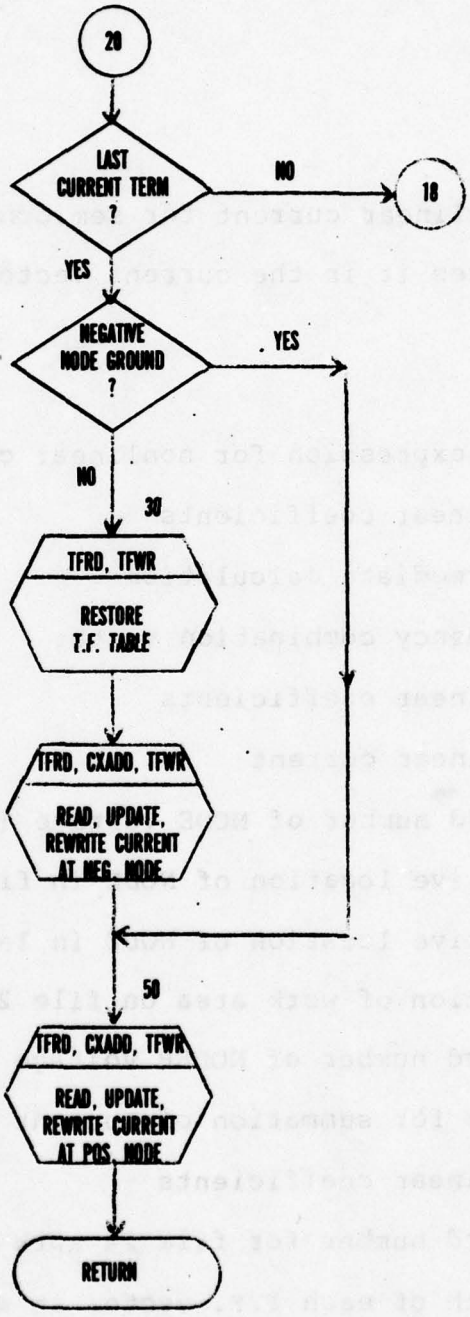
c) If (S.LE.I), go to Step 2. Otherwise coefficient expression complete.

Then the transfer function table is restored by reading the transfer functions saved in the scratch area on file 24 and rewriting them to the original locations in the table. The procedure is similar to that employed for modifying the transfer function table.

The nonlinear currents are placed in the current vector on file 25 by subtracting I2 from the current at NODE+1, subtracting IP from the current at NODE+2, and adding I2 and IP to the current at NODE+3. The current elements for the sequential vacuum pentode nodes, beginning at I=2*NODE-1, are read from file 25 by subroutine TFRD and stored in X, X1, X2, and X3. I2 and IP are subtracted from X1 and X2 respectively by subroutine CXSUB. I2 and IP are added to X3 by subroutine CXADD. The updated current vector elements are rewritten to their original locations by subroutine TFWR and subroutine CURPT returns to the calling program.


CURSD
2
5
5
CXADD
CXSUB
FNCTN
DATARD
TFRD
TFWR


START
DATARD
READ
DATA RECORD
FROM FILE 20
NEGATIVE
NODE GROUND
?
YES
NO
5
INITIALIZE
T.F. TABLE
MODIFICATION
TFRD, CXSUB, TFWR
READ, SAVE,
MODIFY, REWRITE
NEXT T.F. VECTOR
10
LAST
VECTOR
MODIFIED
?
NO
YES
15
INITIALIZE
NONLINEAR
CURRENT
16
FNCTN
CALCULATE
NEXT
CURRENT TERM
ACCUMULATE
CURRENT
TERMS
20


20
LAST CURRENT TERM ?
NO
18
YES
NEGATIVE NODE GROUND ?
YES
NO
30
TFRD, TFWR
RESTORE T.F. TABLE
TFRD, CXADD, TFWR
READ, UPDATE, REWRITE CURRENT AT NEG. NODE
50
TFRD, CXADD, TFWR
READ, UPDATE, REWRITE CURRENT AT POS. NODE
RETURN

NAME: CURSD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear current for semiconductor diode and places it in the current vector

VARIABLES:

A	= T.F. expression for nonlinear current
C2	= Nonlinear coefficients
D	= Intermediate calculation
FREQ	= Frequency combination vector
GAMMA	= Nonlinear coefficients
HKC	= Nonlinear current
I	= Record number of NODE voltage (current)
I1	= Relative location of NODE in first T.F.
I2	= Relative location of NODE in last T.F.
I3	= Location of work area on file 24
J	= Record number of NODEB voltage
K	= Index for summation of current calculation
KS	= Nonlinear coefficients
L	= Record number for file 24 work area
LT	= Length of each T.F. vector in words
NODE	= Positive node of connection
NODEB	= Negative node of connection

NORDR = Order of analysis

S = Angular frequency

X = NODE voltage (current)

Y = NODEP voltage (current)

SUBROUTINES CALLED:

CXADD CXSUB DATARD FNCTN

TFRD FTWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURSD calculates the nonlinear current for the semiconductor diode and places it in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear current to be calculated, and S, the angular frequency. The nonlinear current is a function of the device coefficients and parameters calculated in Phase 1 and the node voltages derived from the lower order transfer functions stored on file 24. The positive and negative nodes of connection, NOD and NODEP, are stored in the semiconductor diode driver record i global common.

First the semiconductor diode data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Then the node voltages for the nonlinear current calculations are computed by replacing the transfer function at

NODE with the difference between the transfer functions at NODE and NODEB for every vector in the table. This "modified transfer function table" serves as the basis of the nonlinear current calculations. If the negative node of connection is ground(NODEB =Ø), modification of the transfer function table is by-passed.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the circuit and occupies two decimal storage locations. The transfer functions are stored in the vectors according to increasing node numbers. This method of storage allows the transfer functions to be accessed and modified by the following iterative procedure:

STEP 1: Initialization

a) Length of each T.F. vector: LT=2*ORDER

b) Location of NODE in first vector: I1=2*NODE-1

c) Location of NODE in last vector: $I2=I1+(2^{NORDR}-3)*LT$

d) Origin of scratch area on file 24: $I=2^{NORDR}*LT$

e) Set index at first vector: I=I1

STEP 2: Access and Modify Next Vector

a) Read transfer function from I, store in X.

b) Write X to scratch area at L, increment L=L+2

c) Locate NODEB in vector: J=I+(NODEB-NODE)*2

d) Read transfer function from J, store in Y.

e) Calculate X=X-Y and write to file 24 at I.

STEP 3: Increment and Test Index

a) Increment I=I+LT for next vector.

b) If (I.LE.I2), more vectors remain to be modified. Go to Step 2.

c) Otherwise modified transfer function table complete.

The nonlinear current of order NORDR for the semiconductor diode is defined as:

$$HKG = \sum_{K=2}^{NORDR} (KS(K) * D) * F_K^{NORDR}(\overline{NODE})$$

where:

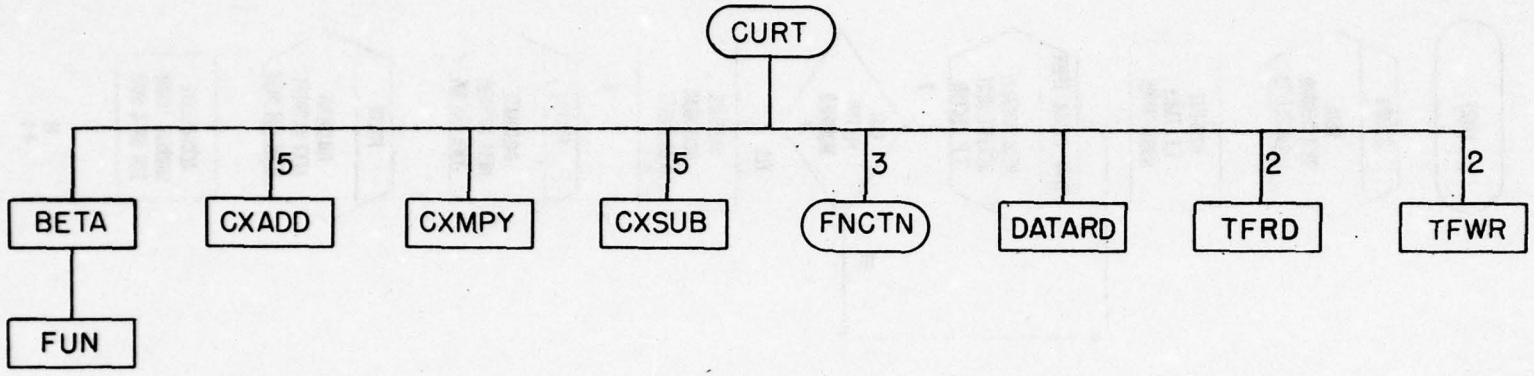
D=S*(GAMMA(K)+C2(2)*KS(K-1)/K)

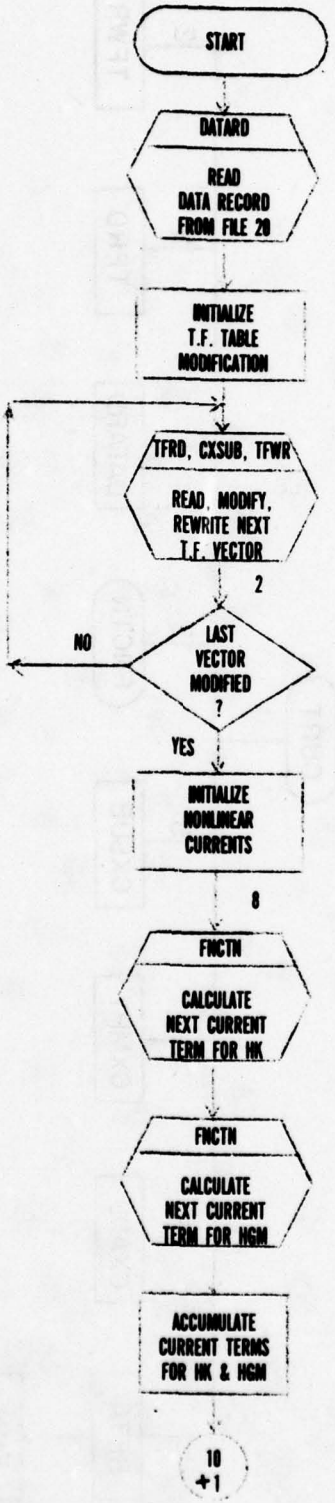
Subroutine CURSD initializes HKG to zero, then each symmetric transfer function expression $F_K^{NORDR}(\overline{NODE})$ for K=2,3,... NORDR is computed by subroutine FNCTN. After the intermediate

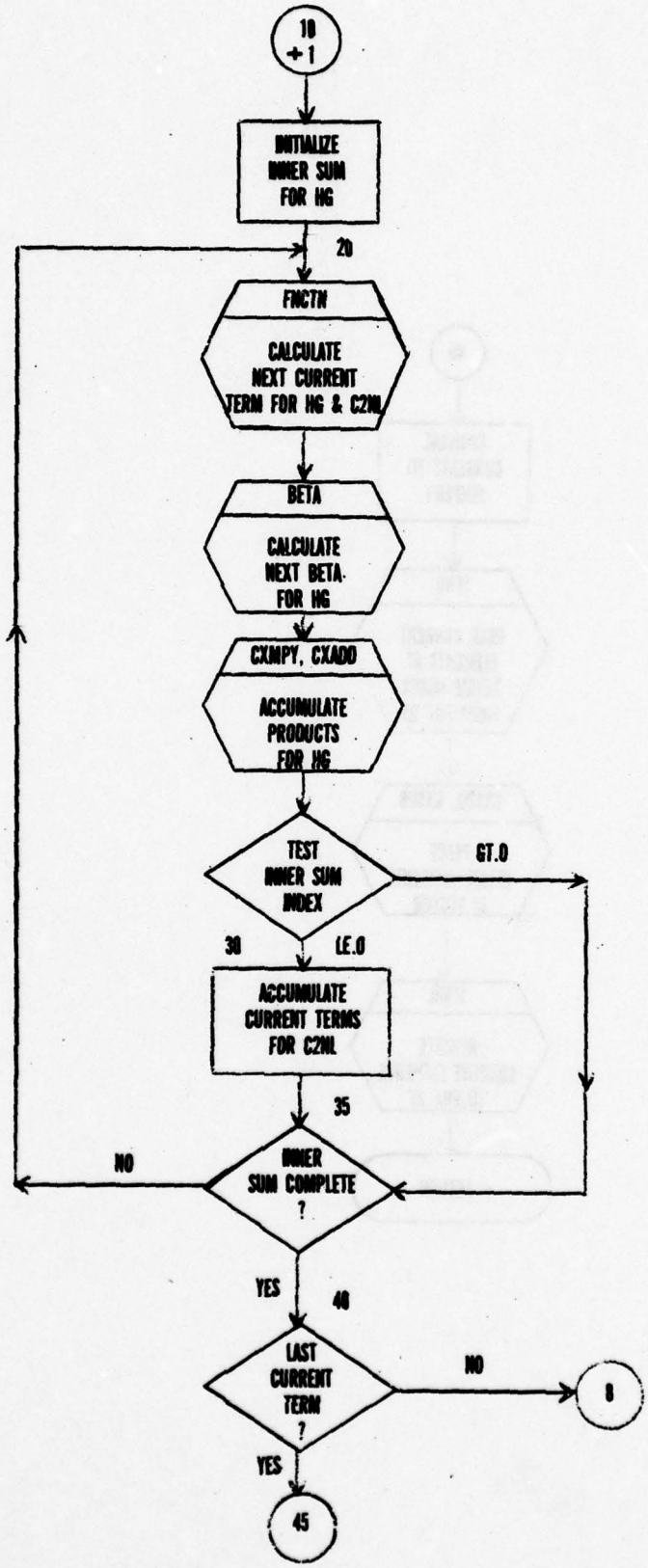
calculation is performed for D, the product is formed and accumulated in HKG.

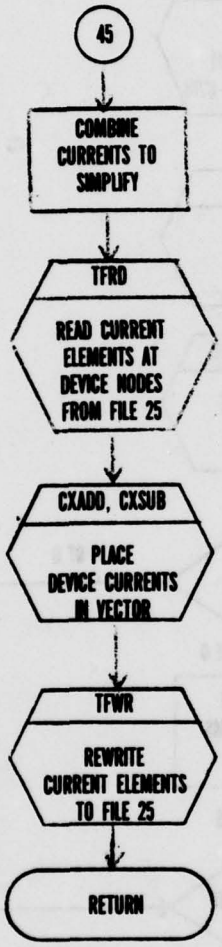
Then the transfer function table is restored by reading the transfer functions saved in the scratch area on file 24 and rewriting them to their original locations in the table. The procedure is similar to that employed for modifying the transfer function table. If the negative node of connection is ground (NODEB=Ø), the restore procedure is by-passed.

The nonlinear current is placed in the current vector on file 25 by adding it to both the current at NODEB (for NODEB≠Ø) and the current at NODE. The NODEB current, located at I=2*NODEB-1, is read from file 25 by subroutine TFRD and stored in X. HKG is added to X by subroutine CXADD and the updated NODEB current is rewritten to its original location by subroutine TFWR. The procedure is repeated for the NODE current located at I=2*NODE-1, and subroutine CURSD returns to the calling program.


CURT
BETA
FUN
5
CXADD
CXMPY
5
CXSUB
3
FNCTN
DATARD
2
TFRD
2
TFWR


START
DATARD
READ DATA RECORD FROM FILE 20
INITIALIZE T.F. TABLE MODIFICATION
TFRD, CXSUB, TFWR
READ, MODIFY, REWRITE NEXT T.F. VECTOR
2
LAST VECTOR MODIFIED ?
NO
YES
INITIALIZE NONLINEAR CURRENTS
8
FNCTN
CALCULATE NEXT CURRENT TERM FOR HK
FNCTN
CALCULATE NEXT CURRENT TERM FOR HGM
ACCUMULATE CURRENT TERMS FOR HK & HGM
10
+1


10
+1
INITIALIZE
INNER SUM
FOR HG
20
FNCTN
CALCULATE
NEXT CURRENT
TERM FOR HG & C2NL
BETA
CALCULATE
NEXT BETA
FOR HG
CXMPY, CXADD
ACCUMULATE
PRODUCTS
FOR HG
TEST
INNER SUM
INDEX
GT.0
30
LE.0
ACCUMULATE
CURRENT TERMS
FOR C2NL
35
NO
INNER
SUM COMPLETE
?
YES
40
LAST
CURRENT
TERM
?
NO
8
YES
45


45
COMBINE CURRENTS TO SIMPLIFY
TFRD
READ CURRENT ELEMENTS AT DEVICE NODES FROM FILE 25
CXADD, CXSUB
PLACE DEVICE CURRENTS IN VECTOR
TFWR
REWRITE CURRENT ELEMENTS TO FILE 25
RETURN

NAME: CURT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear currents for bipolar junction transistor and places them in current vector

VARIABLES:

A	= T.F. expression for nonlinear current
ALPHA	= Nonlinear coefficients
B	= Intermediate calculation for HG
C	= Intermediate calculation for HG
C2NL	= Nonlinear current
D	= Intermediate calculation for C2NL
FREQ	= Frequency combination vector
GAMMA	= Nonlinear coefficients
HC	= Nonlinear coefficients
HG	= Nonlinear current
HGM	= Nonlinear current
HK	= Nonlinear current
HM	= Nonlinear coefficients
ICHMO	= Nonlinear parameter
I	= Record number of NODE voltage (current)
I1	= Relative location of NODE in first T.F.
I2	= Relative location of NODE in last T.F.

I3 = Location of work area on file 24

K = Index for outer summation of current calculation

L = Record number for file. 24 work area

KS = Nonlinear coefficients

LT = Length of each T.F. vector in words

M = Index for inner summation of current calculation.

NODE = Base node of connection

NORDR = Order of analysis

S = Angular frequency

SCO = Nonlinear parameter

SIGN = Sign of HGM current terms

SNCS = Nonlinear parameter

X = Base node voltage (current)

X1 = (Base+1) voltage (current)

X2 = (Base+2) voltage (current)

X3 = (Base+3) voltage (current)

SUBROUTINES CALLED

BETA CXADD CXMPY CXSUB

DATARD FNCTN TFRD

TFWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURT calculates the nonlinear currents for the bipolar junction transistor and places them in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear currents to be calculated, and S, the angular frequency. The nonlinear currents are functions of the device parameters and coefficients calculated in Phase 1 and the node voltages derived from the lower order transfer functions stored on file 24. The base node of connection, NODE, is stored in the transistor driver record in global common. The remaining nodes are sequentially numbered by the system as NODE+1, NODE+2, and NODE+3.

First the transistor data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Then the node voltages for the nonlinear current calculations are computed for each vector in the transfer function table as follows: The transfer function at NODE is replaced by the difference between the transfer functions at NODE+2 and NODE; the transfer function at NODE+2 is replaced by the difference between the transfer functions at NODE+2 and NODE+1; and the transfer function at NODE+1 is replaced by the difference between the transfer functions at NODE+1 and NODE+3. This "modified transfer function table" serves as the basis for the nonlinear current calculations.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the

circuit and occupies two decimal storage locations. The transfer functions are stored in the vectors according to increasing node numbers. Because the transistor nodes are numbered sequentially, the corresponding transfer functions occupy successive elements in each vector. This method of storage allows the transfer functions to be accessed and modified by the following iterative procedure:

STEP 1: Initialization

a) Length of each T.F. vector: LT=2*ORDER

b) Location of NODE in first vector: I1=2*NODE-1

c) Location of NODE in last vector: $I2=I1+(2^{NORDR}-3)*LT$

d) Origin of scratch area on file 24: $L = 2^{NORDR} * LT$

e) Set index at first vector: I=I1

STEP 2: Access and Modify Next Vector

a) Read transfer functions for sequential device nodes beginning at I, store in X, X1, X2, and X3

b) Write device voltages to scratch area at L, increment L=L+2

c) Calculate X=X2-X, X2=X2-X1, and X1=X1-X3

d) Write updated transfer function to file 24 beginning at I

STEP 3· Increment and Test Index

a) Increment I=I+LT

b) If (I.LE.I2), more vectors remain to be modified. Go to Step 2.

c) Otherwise modified transfer function table complete.

The nonlinear currents of order NORDR for the bipolar junction transistor are defined as:

$$HK = \sum_{K=2}^{NORDR} KS(K) * F_K^{NORDR} \quad (\overline{NODE + 2})$$

$$HGM = \sum_{K=2}^{NORDR} S * GAMMA(K) * F_K^{NORDR} \quad (\overline{NODE + 3})$$

$$HG = \sum_{K=2}^{NORDR} \sum_{M=0}^{K} B_M^K * F_K^{NORDR} Q_M^K \quad (\overline{NODE + 2, NODE + 1})$$

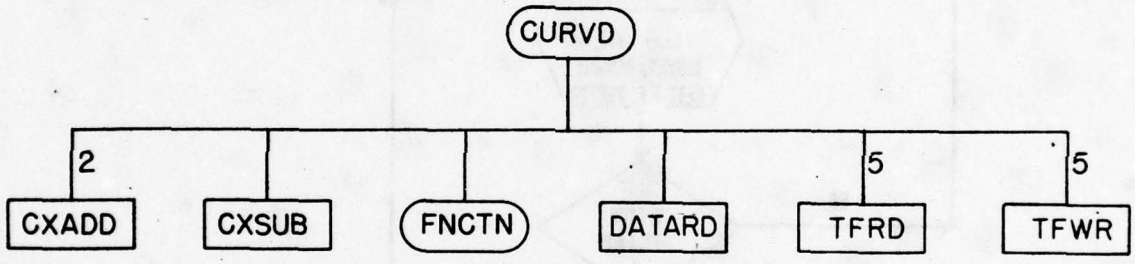
$$C2NL = \sum_{K=2}^{NORDR} D * F_K^{NORDR} \quad (\overline{NODE + 1})$$

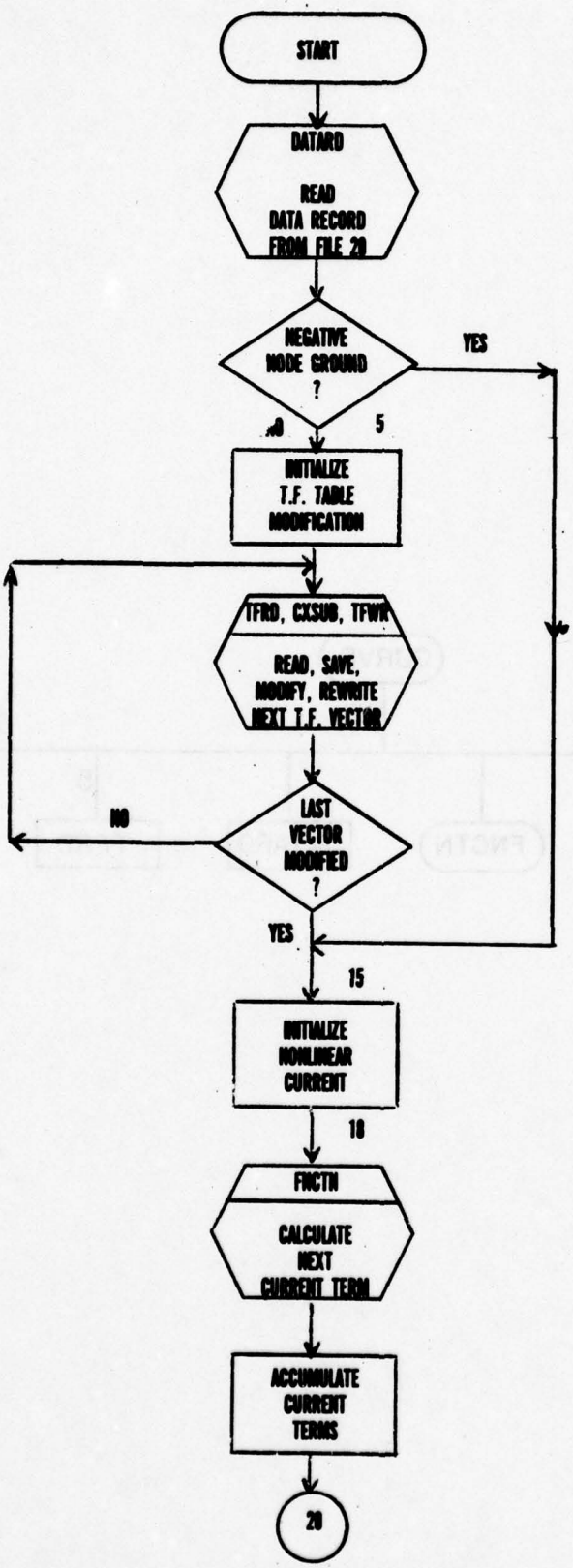
where D=S*HC(2)*KS(K-1)/K. Subroutine CURT initializes HK, HGM, HG, and C2NL to zero. Then the symmetric transfer function expressions $F_K^{NORDR}(\overline{NODE+2})$ and $F_K^{NORDR}(\overline{NODE+3})$ for K=2,3,...NORDR are computed by subroutine FNCTN, multiplied by the appropriate coefficients KS(K) or S*GAMMA(K), and accumulated in HK and HGM respectively. Then the symmetric transfer function expression $F_K^{NORDR}\, Q_M^K(\overline{NODE+2,\ NODE+1})$ for K=2,3,... NORDR and M=0,1,...K is computed by subroutine FNCTN. Subroutine BETA is called to compute B_M^K and the product is formed and accumulated in HG. The symmetric transfer function expression $F_K^{NORDR}(\overline{NODE+1})$ is identical to the expression for HG evaluated at M=0. The intermediate calculation for D is performed and the product is formed and accumulated in C2NL.

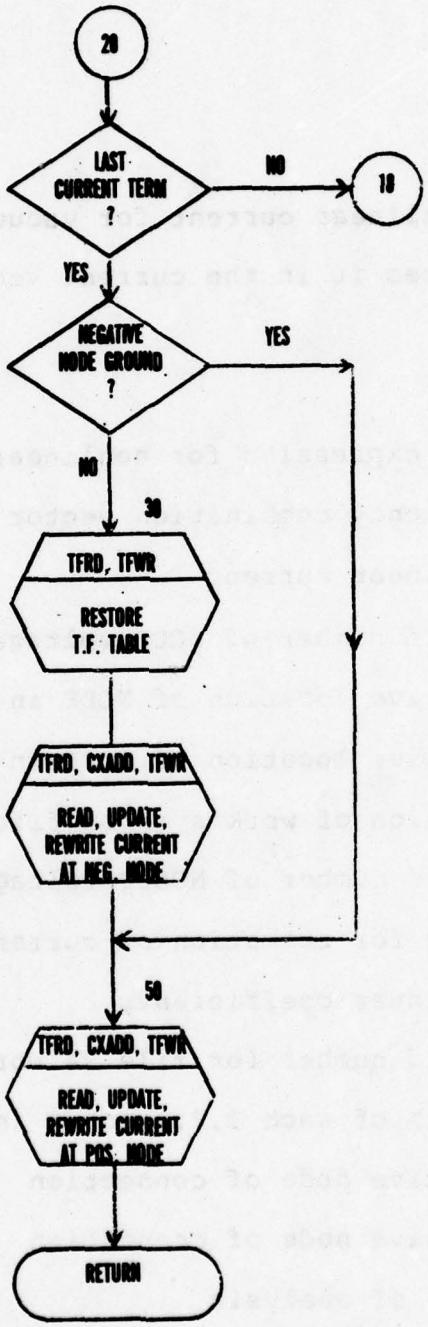
Then the transfer function table is restored by reading the transfer functions saved in the scratch area on file 24 and rewriting them to the original locations in the table. The procedure is similar to that employed for modifying the transfer function table.

The nonlinear currents are placed in the current vector on file 25 by adding HG and HGM to the current at NODE+1, adding HK and C2NL to the current at NODE+3, subtracting HK and C2NL from the current at NODE+1, and subtracting HG and HCM from the current at NODE+2. The current elements for the sequential transistor nodes, beginning at I=2*NODE-1, are read from file 25 by subroutine TFRD and stored in X, X1, X2, and X3. HG and HGM are added to X1 and HK and C2NL added to X3 by subroutine CXADD.

HK and C2NL are subtracted from X1 and HG and HGM are subtracted from X2 by subroutine CXSUB. The updated current vector elements are rewritten to their original locations by subroutine TFWR and subroutine CURT returns to the calling program.


CURVD
2
CXADD
CXSUB
FNCTN
DATARD
5
TFRD
5
TFWR


START
DATARD
READ
DATA RECORD
FROM FILE 20
NEGATIVE
NODE GROUND
?
YES
.0
5
INITIALIZE
T.F. TABLE
MODIFICATION
TFRD, CXSUB, TFWR
READ, SAVE,
MODIFY, REWRITE
NEXT T.F. VECTOR
LAST
VECTOR
MODIFIED
?
NO
YES
15
INITIALIZE
NONLINEAR
CURRENT
10
FRCTN
CALCULATE
NEXT
CURRENT TERM
ACCUMULATE
CURRENT
TERMS
20


20
LAST CURRENT TERM ?
NO
18
YES
NEGATIVE MODE GROUND ?
YES
NO
30
TFRD, TFWR
RESTORE T.F. TABLE
TFRD, CXADO, TFWR
READ, UPDATE, REWRITE CURRENT AT NEG. MODE
50
TFRD, CXADO, TFWR
READ, UPDATE, REWRITE CURRENT AT POS. MODE
RETURN

NAME: CURVD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear current for vacuum diode and places it in the current vector

VARIABLES:

A	= T.F. expression for nonlinear current
FREQ	= Frequency combination vector
HK	= Nonlinear current
I	= Record number of NODE voltage (current)
I1	= Relative location of NODE in first T.F.
I2	= Relative location of NODE in last T.F.
I3	= Location of work area on file 24
J	= Record number of NODEB voltage
K	= Index for summation of current calculation
KS	= Nonlinear coefficients
L	= Record number for file 24 work area
LT	= Length of each T.F. vector in words
NODE	= Positive node of connection
NODEB	= Negative node of connection
NORDR	= Order of analysis
S	= Angular frequency
Y	= NODE voltage (current)

Y = NODEB voltage (current)

SUBROUTINES CALLED:

CXADD CXSUB DATARD FNCTN TFRD

TFWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURVD calculates the nonlinear current for the vacuum diode and places it in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear current to be calculated, and S, the angular frequency. The nonlinear current is a function of the device coefficients calculated in Phase 1 and the node voltage derived from the lower order transfer functions stored on file 24. The positive and negative nodes of connection NODE and NODEB are stored in the vacuum diode driver record in global common.

First the vacuum diode data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Then the node voltages for the nonlinear current calculations are computed by replacing the transfer function at NODE with the difference between the transfer functions at NODE and NODEB for every vector in the table. This "modified transfer function table" serves as the basis for the nonlinear current calculations. If the negative node of connection is ground

(NODEB=0), modification of the transfer function table is by-passed.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the circuit and occupies two decimal storage locations. The transfer functions are stored in the vectors according to increasing node numbers. This method of storage allows the transfer functions to be accessed and modified by the following iterative procedure:

STEP 1: Initialization

a) Length of each T.F. vector: LT=2*ORDER

b) Location of NODE in first vector: I1=2*NODE-1

c) Location of NODE in last vector: $I2=I1+(2^{NORDR}-3)*LT$

d) Origin of scratch area on file 24: $L=2^{NORDR}*LT$

e) Set index at first vector: I=I1

STEP 2: Access and Modify Next Vector

a) Read transfer function from I, store in X.

b) Write X to scratch area at L, increment L=L+2.

c) Locate NODEB in vector: J=I+(NODEB-NODE)*2.

d) Read transfer function from J, store in Y.

e) Calculate X=X-Y and write to file 24 at I.

STEP 3: Increment and Test index

a) Increment I=I+LT for next vector.

b) If (I.LE.I2), more vectors remain to be modified.
Go to Step 2.

c) Otherwise modified transfer function table complete.

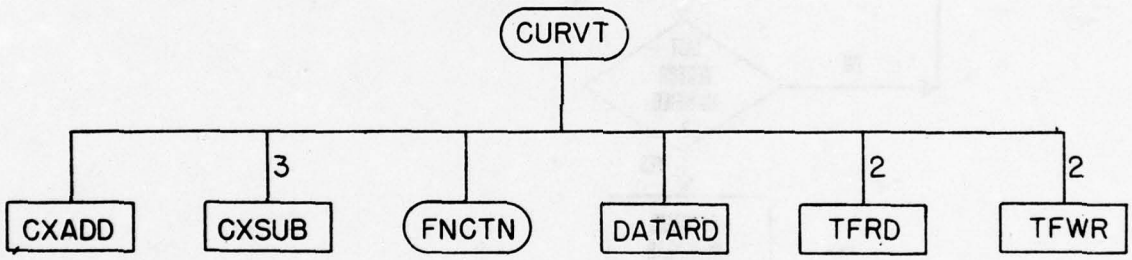
The nonlinear current of order NORDR for the vacuum diode is defined as:

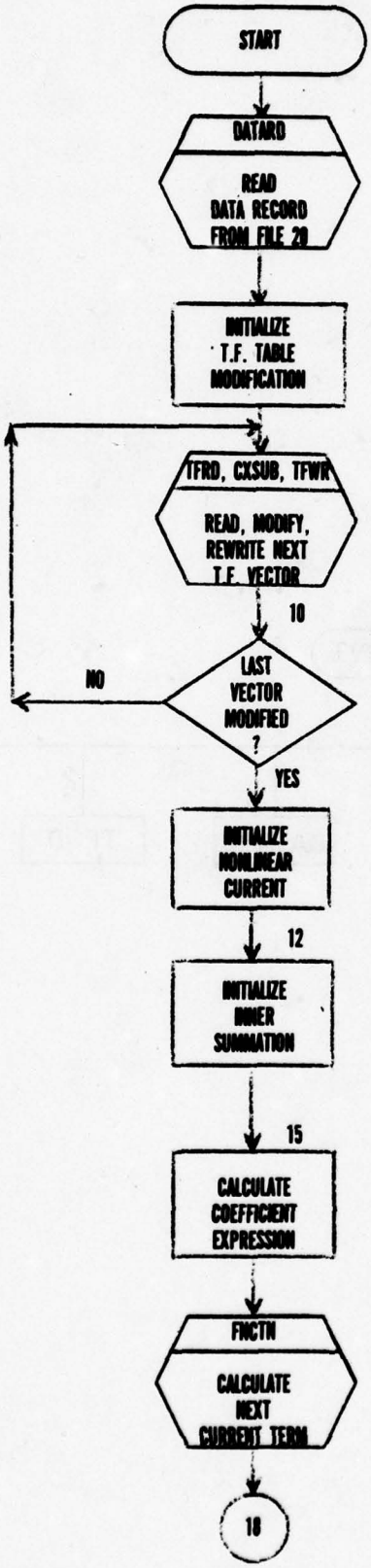
$$HK = \sum_{K=2}^{NORDR} KS(K) * F_K^{NORDR}(\overline{NODE})$$

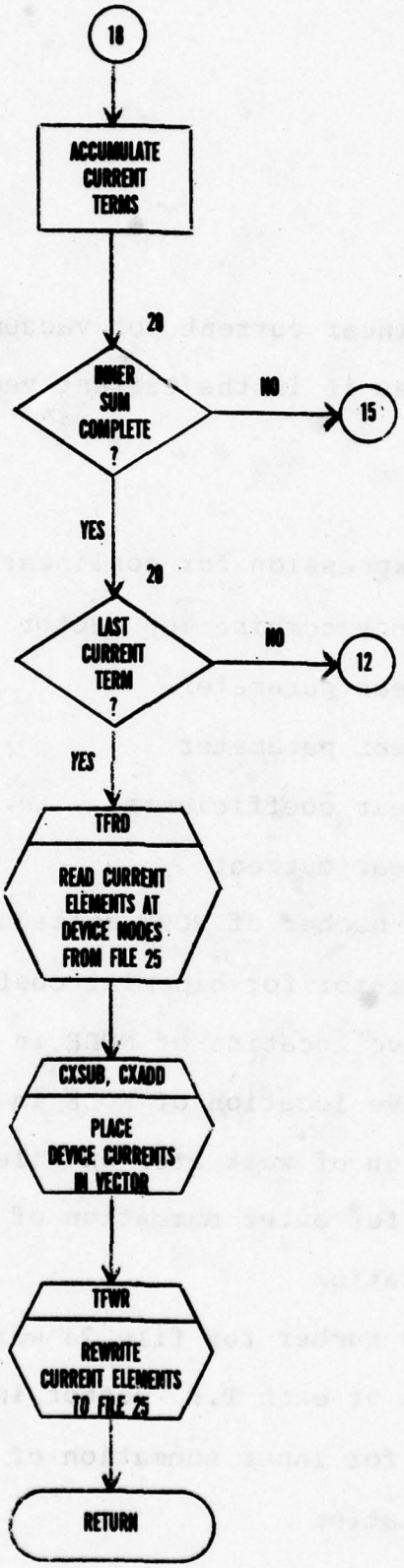
Subroutine CURVD initializes HK to zero, then each symmetric transfer function expression $F_K^{NORDR}(\overline{NODE})$ for K=2,3,...NORDR is computed by subroutine FNCTN, multiplied by the appropriate coefficient KS(K), and accumulated in HK.

Then the transfer function table is restored by reading the transfer functions saved in the scratch area on file 24 and rewriting them to their original locations in the table. The procedure is similar to that employed for modifying the transfer function table. If the negative node of connection is ground (NODEB=0), the restore procedure is by-passed.

The nonlinear current is placed in the current vector on file 25 by subtracting it from the current at NODEB (for NODEB ≠0) and adding it to the current at NODE. The NODEB current, located at I=2*NODEB-1, is read from file 25 by subroutine TFRD and stored in X. HK is subtracted from X by subroutine CXSUB and the updated NODEB current is rewritten to its original location by TFWR. The procedure is repeated for the NODE current, except that NODE is located at I=2*NODE-1 and HK is added to X by subroutine CXADD. Subroutine CURVD then returns to the calling program.


CURVT
3
2
2
CXADD
CXSUB
FNCTN
DATARD
TFRD
TFWR


START
DATARD
READ DATA RECORD FROM FILE 20
INITIALIZE T.F. TABLE MODIFICATION
TFRD, CXSUB, TFWR
READ, MODIFY, REWRITE NEXT T.F. VECTOR
10
LAST VECTOR MODIFIED ?
NO
YES
INITIALIZE NONLINEAR CURRENT
12
INITIALIZE INNER SUMMATION
15
CALCULATE COEFFICIENT EXPRESSION
FNCTN
CALCULATE NEXT CURRENT TERM
18


18
ACCUMULATE CURRENT TERMS
20
INNER SUM COMPLETE ?
NO
15
YES
20
LAST CURRENT TERM ?
NO
12
YES
TFRD
READ CURRENT ELEMENTS AT DEVICE NODES FROM FILE 25
CXSUB, CXADD
PLACE DEVICE CURRENTS IN VECTOR
TFWR
REWRITE CURRENT ELEMENTS TO FILE 25
RETURN

NAME: CURVT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates nonlinear current for vacuum triode and places it in the current vector

VARIABLES:

A	= T.F. expression for nonlinear current
FREQ	= Frequency combination vector
G	= Nonlinear parameter
GP	= Nonlinear parameter
H	= Nonlinear coefficients
HG	= Nonlinear current
I	= Record number of NODE voltage (current)
IFACT	= Accumulator for binomial coefficient
I1	= Relative location of NODE in first T.F.
I2	= Relative location of NODE in last T.F.
I3	= Location of work area on file 24
K	= Index for outer summation of current calculation
L	= Record number for file 24 work area
LT	= Length of each T.F. vector in words
M	= Index for inner summation of current calculation

MU = Input value

NODE = Base node of connection

NORDR = Order of analysis

Q = Intermediate calculation

S = Angular frequency

X = Base node voltage (current)

X1 = (Base+1) voltage (current)

X2 = (Base+2) voltage (current)

SUBROUTINES CALLED:

CXADD CXSUB DATARD FNCTN

TFRD TFWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine CURVT calculates the nonlinear current for the vacuum triode and places it in the current vector. Arguments transmitted to the subroutine are: NORDR, the order of the nonlinear current to be calculated, and S, the angular frequency. The nonlinear current is a function of the device parameters and coefficients calculated in Phase 1 and the node voltages derived from the lower order transfer functions stored on file 24. The base node of connection, NODE, is stored in the vacuum triode driver record in global common. The remaining nodes are sequentially numbered by the system as NODE+1 and NODE+2.

First the vacuum triode data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Then the node voltages for the nonlinear current calculations are computed as follows: the transfer function at NODE is replaced by the difference between the transfer functions at NODE and NODE+2 for every vector in the table and the transfer function at NODE+1 is replaced by the difference between the transfer functions at NODE+1 and NODE+2 divided by the input parameter MU for every vector in the table. This "modified transfer function table" serves as the basis for the nonlinear current calculations.

The transfer function table consists of $2^{NORDR}-2$ contiguous vectors beginning at record number 1 on file 24. Each complex vector element is the transfer function at some node in the circuit and occupies two decimal storage locations. The transfer functions are stored in the vectors according to increasing node numbers. Because the vacuum triode nodes are numbered sequentially, the corresponding transfer functions occupy successive elements in each vector. This method of storage allows the transfer functions to be accessed and modified by the following iterative procedure:

STEP 1: Initialization

a) Length of each T.F. vector: LT=2*ORDER

b) Location of NODE in first vector: I1=2*NODE-1

c) Location of NODE in last vector: $I2=I1+(2^{NORDR}-3)*LT$

d) Origin of scratch area on file 24: $L = 2^{NORDR} * LT$

e) Set index at first vector: I=I1

STEP 2: Access and Modify Next Vector

a) Read transfer functions for sequential device nodes beginning at I, store in X, X1, and X2.

b) Write device voltages to scratch area at L, increment L=L+2

c) Calculate X=X-X2 and X1=(X1-X2)/MU

d) Write updated transfer functions to file 24 beginning at I

STEP 3: Increment and Test Index

a) Increment I=I+LT

b) If (I.LE.LT), more vectors remain to be modified. Go to Step 2.

c) Otherwise modified transfer function table complete.

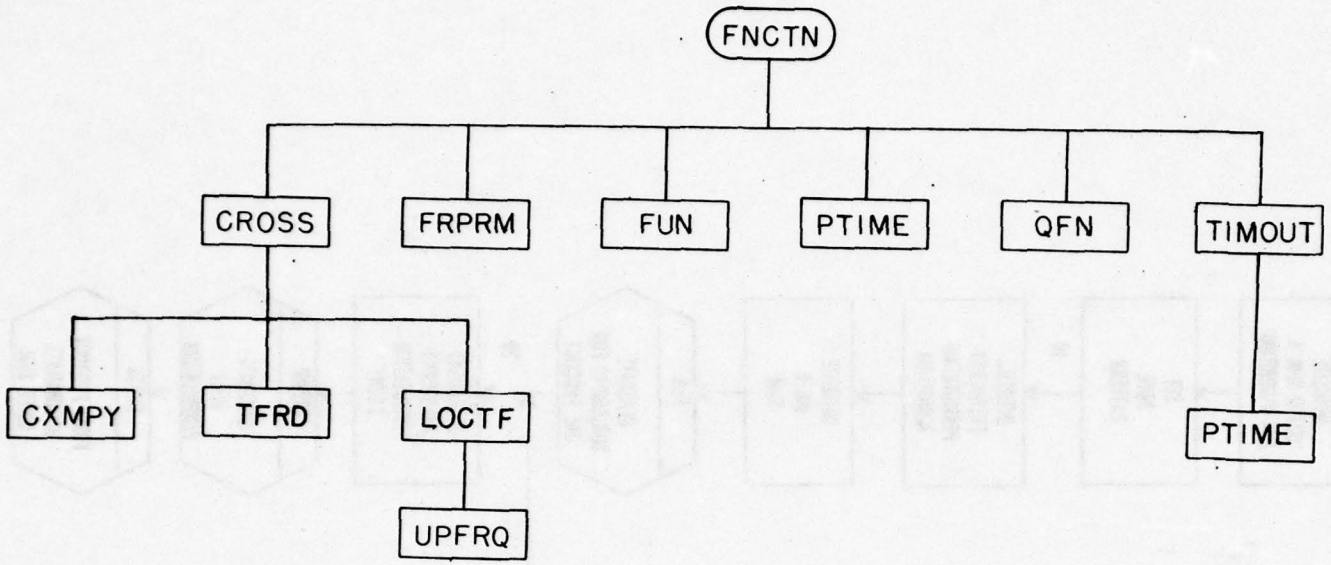
The nonlinear current of order NORDR for the vacuum triode is defined as:

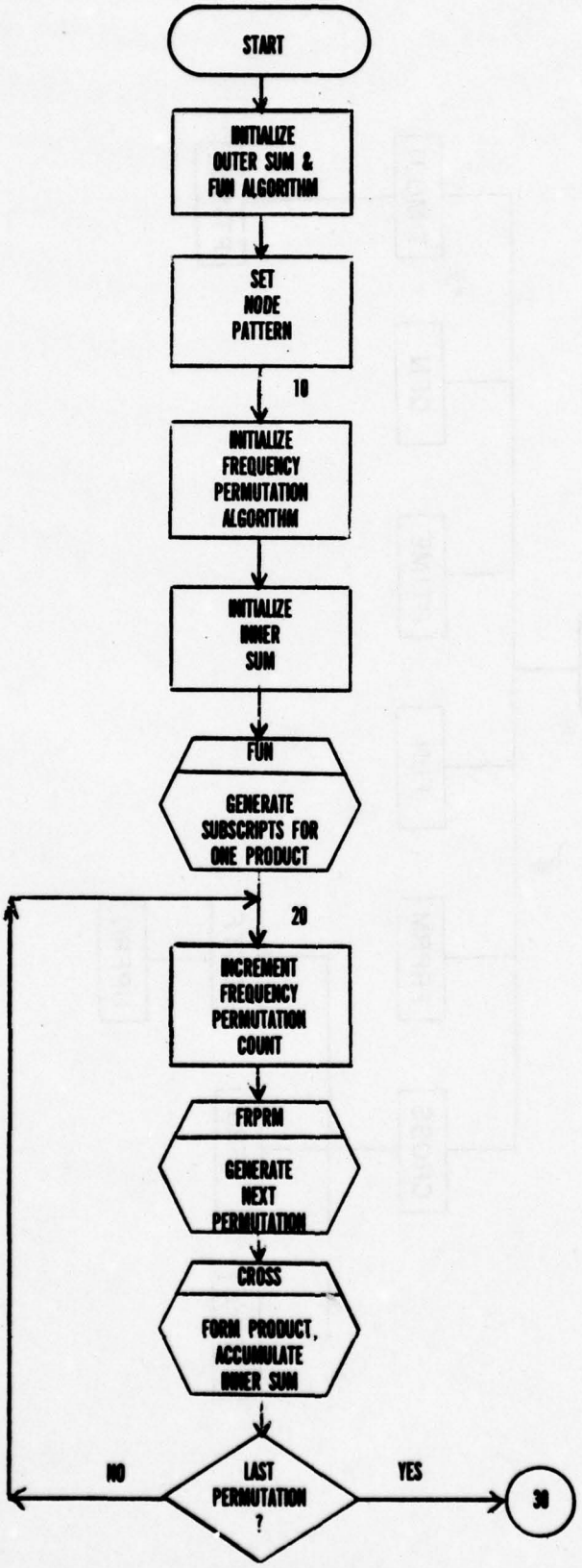
$$HG = \sum_{K=2}^{NORDR} \sum_{M=0}^{K} D_{M,K} * F_K^{NORDR} \; Q_M^K \; \overline{[NODE + 1, \; NODE + 2]}$$

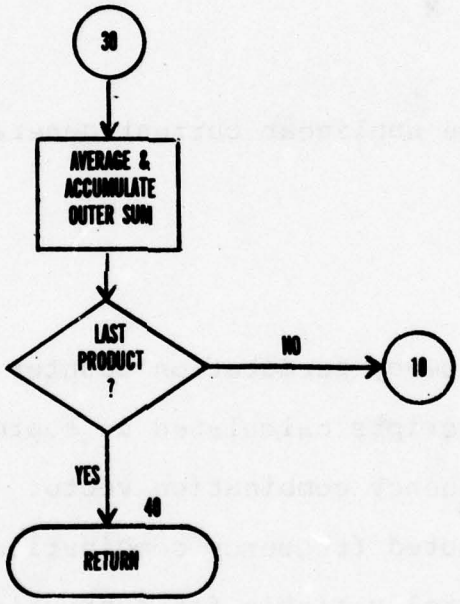
where $D_{M,K} = \binom{K}{M}$ (K-M)*GP*H(K)/K+G*H(K+1). Subroutine CURVT initializes HG to zero, then each symmetric transfer function expression $F_K^{NORDR} \; Q_M^K \; \overline{[NODE+1, \; NODE+2]}$ for K=2,3,... NORDR and M=Ø,1...K is computed by subroutine FNCTN. After the intermediate calculations are performed for $D_{M,K}$, the binomial coefficient $\binom{K}{M}$ is accumulated in IFACT, and the product is formed and accumulated in HG.

Then the transfer function table is restored by reading the transfer functions saved in the scratch area on file 24 and rewriting them to the original locations in the table. The procedure is similar to that employed for modifying the transfer function table.

The nonlinear current is placed in the current vector on file 25 by subtracting it from the current at NODE+1 and adding it to the current at NODE+2. The current elements for the sequential vacuum triode nodes, beginning at I=2*NODE-1, are read from file 25 by subroutine TFRD and stored in X, X1, and X2. HG is subtracted from X1 by subroutine CXSUB and added to X2 by subroutine CXADD. The updated current elements are rewritten to their original locations by subroutine TFWR and subroutine CURVT returns to the calling program.


FNCTN
CROSS
FRPRM
FUN
PTIME
QFN
TIMOUT
CXMPY
TFRD
LOCTF
UPFRQ
PTIME


START
INITIALIZE OUTER SUM & FUN ALGORITHM
SET MODE PATTERN
10
INITIALIZE FREQUENCY PERMUTATION ALGORITHM
INITIALIZE INNER SUM
FUN
GENERATE SUBSCRIPTS FOR ONE PRODUCT
20
INCREMENT FREQUENCY PERMUTATION COUNT
FRPRM
GENERATE NEXT PERMUTATION
CROSS
FORM PRODUCT, ACCUMULATE INNER SUM
LAST PERMUTATION ?
NO
YES
30


30
AVERAGE &
ACCUMULATE
OUTER SUM
LAST
PRODUCT
?
NO
10
YES
40
RETURN

NAME: FNCTN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates the symmetrized transfer function expression

$$F_K^{N\ K}\ Q\ (\overline{I1,I2})_M$$

for use by the nonlinear current generator subroutines

VARIABLES:

CNT = Frequency permutation counter

FNK = Subscripts calculated in subroutine FUN

FREQ = Frequency combination vector

FREQC = Permuted frequency combination vector

ISWF = Control variable for subroutine FPPRM

ISWT = Control variable for subroutine FUN

K = Length of FNK vector

M = Number of nodes I2 in node pattern

N = Order of T.F. expression

Q = Node pattern of T.F. expression

U = Location of nonlinearity relative to base node

V = Location of dependence relative to base node

XX = Calculated symmetrized T.F. expression

YY = Sum of T.F. products calculated in subroutine CROSS

SUBROUTINES CALLED:

CROSS FRPRM FUN PTIME QFN

TIMOUT

CALLING PROGRAMS:

CURJF CURNC CURNDS CURSD CURT

CURVD CURVT

DESCRIPTION:

For given N, K and M subroutine FNCTN calculates the expression $F_K^N \overline{[Q_M^K (U,V)]}$, which is a sum of symmetric products of voltages from the modified transfer function table used in calculating nonlinear currents during Phase 3.

A nonlinear current is a function of the voltage across the nonlinearity as well as the voltage at some other node in the circuit. A nonlinear current of order N can be expressed as a power series in voltage at two nodes, U and V:

$$I_N (f_1, f_2, \ldots f_N) = \sum_{K=2}^{N} \sum_{M=0}^{K} KS_{K,M} F_K^N \overline{[Q_M^K (U,V)]}$$

The KS's are functions of the Taylor series coefficients calculated in Phase 1. $\overline{F_K^N\ [Q_M^K\ (U, V)]}$ is a sum of products of K lower order voltages from the modified transfer function table. Each product has the following general form:

$$\underbrace{U_{p_1}\ U_{p_2}\ \ldots U_{p_{K-M}}}_{\text{K-M Voltages}} \qquad \underbrace{V_{p_{K-M+1}}\ \ldots V_{p_K}}_{\text{M Voltages}}$$

The function $Q_M^K\ (U, V)$ determines the pattern of node voltages U and V so that each transfer function product contains K voltages, where the first K-M voltages are from node U and the remaining M voltages are from node V. The function F_K^N determines the order P_i of each transfer function in the product so that $p_1 + p_2 + \ldots p_K = N$. The overbar depicts the symmetry of the product, achieved by averaging over the N! possible permutations of the N input frequencies.

For example, a third order nonlinear current is given by:

$$I3(f_1,f_2,f_3) = \sum_{K=2}^{3} \sum_{M=0}^{K} KS_{K,M}\ F_K^3\ [\overline{Q_M^K\ (u,v)}]$$

$$= \sum_{K=2}^{3} KS_{K,0}\ F_K^3\ \overline{[Q_0^K\ (u,v)]} + KS_{K,1}\ F_K^3\ \overline{[Q_1^K\ (u,v)]} + \ldots$$

$$+ KS_{K,K}\ F_K^3\ \overline{[Q_K^K\ (u,v)]}$$

Rather than expanding the expression completely, consider the current component at K=2:

$$KS_{2,0}\, F_2^3\, \overline{[Q_0^2\ (u,v)\ ]} + KS_{2,1}\, F_2^3\, \overline{[Q_1^2\ (u,v)]} + KS_{2,2}\, F_2^3\, \overline{[Q_2^2\ (u,v)]}$$

Each F_2^3 represents a sum of products of first and second order voltages. The nodes from which the voltages are derived are determined by the Q function where Q_0^2 represents the node pattern UU, Q_1^2 represents VU, and Q_2^2 represents VV. Together the two functions produce the following additive terms:

$$F_2^3\ \overline{[Q_0^2\ (U,V)]} = \overline{U_2 U_1} + \overline{U_1 U_2}$$

$$F_2^3\ \overline{[Q_1^2\ (U,V)]} = \overline{V_2 U_1} + \overline{V_1 U_2}$$

$$F_2^3\ \overline{[Q_2^2\ (U,V)]} = \overline{V_2 V_1} + \overline{V_1 V_2}$$

The symmetry of each product, depicted by the overbar, is achieved by averaging over all the 3!=6 possible permutations of the three input frequencies. Let the six permutations be

represented by:

$$(f_1, f_2, f_3)$$
$$(f_1, f_3, f_2)$$
$$(f_2, f_1, f_3)$$
$$(f_2, f_2, f_1)$$
$$(f_3, f_1, f_2)$$
$$(f_3, f_2, f_1)$$

Then the symmetric product $\overline{U_2U_1}$ is

$$\frac{1}{6}[U(f_1, f_2)U(f_3) + U(f_1, f_3)U(f_2) + U(f_2, f_1)U(f_3) +$$

$$U(f_2, f_3)U(f_1) + U(f_3, f_1)U(f_2) + U(f_3, f_2)U(f_1)]$$

The symmetric product $\overline{U_1U_2}$ is:

$$\frac{1}{6}[U(f_1)U(f_2, f_3) + U(f_1)U(f_3, f_2) + U(f_2)U(f_1, f_3) +$$

$$U(f_2)U(f_3, f_1) + U(f_3)U(f_1, f_2) + U(f_3)U(f_2, f_1)]$$

The symmetric product $\overline{V_2U_1}$ is:

$$\frac{1}{6}[V(f_1, f_2)U(f_3) + V(f_1, f_3)U(f_2) + V(f_2, f_1)U(f_3) +$$

$$V(f_2, f_3)U(f_1) + V(f_3, f_1)U(f_2) + V(f_3, f_2)U(f_1)]$$

The symmetric product $\overline{V_1 U_2}$ is:

$$\frac{1}{6}[V(f_1)U(f_2,f_3) + V(f_1)U(f_3,f_2) + V(f_2)U(f_1,f_3) +$$

$$V(f_2)U(f_3,f_1) + V(f_3)U(f_1,f_2) + V(f_3)U(f_2,f_1)]$$

The symmetric product $\overline{V_2 V_1}$ is:

$$\frac{1}{6}[V(f_1, f_2)V(f_3) + V(f_1, f_3)V(f_2) + V(f_2, f_1)V(f_3) +$$

$$V(f_2, f_3)V(f_1) + V(f_3, f_1)V(f_2) + V(f_3, f_2)V(f_1)]$$

The symmetric product $\overline{\overline{V_1 V_2}}$ is:

$$\frac{1}{6}[V(f_1)V(f_2,f_3) + V(f_1)V(f_3,f_2) + V(f_2)V(f_1,f_3) +$$

$$V(f_2)V(f_3,f_1) + V(f_3)V(f_1,f_2) + V(f_3)V(f_2,f_1)]$$

The nonlinear current component at K=3 is derived in a similar fashion.

$$KS_{3'0}\, F_3^3\, \overline{[Q_0^3(U, V)]} + KS_{3'1}\, F_3^3\, \overline{[Q_1^3(U, V)]} +$$

$$KS_{3'2}\, F_3^3\, \overline{[Q_2^3(U, V)]} + KS_{3'3}\, F_3^3\, \overline{[Q_3^3(U, V)]}$$

where each F represents a product of three first order

voltages. The nodes from which the voltages are derived are determined by the Q function where Q_0^3 represents the node pattern UUU, Q_1^3 represents VUU, Q_2^3 represents VVU, and Q_3^3 represents VVV. Together the two functions produce the following additive terms:

$$F_3^3\ [Q_0^3\ (U, V)] = \overline{U_1 U_1 U_1}$$

$$F_3^3\ [Q_1^3\ (U, V)] = \overline{V_1 U_1 U_1}$$

$$F_3^3\ [Q_2^3\ (U, V)] = \overline{V_1 V_1 U_1}$$

$$F_3^3\ [Q_3^3\ (U, V)] = \overline{V_1 V_1 V_1}$$

Some nonlinear currents are functions of the voltage at a single node. In this case M= Ø and the current expression reduces to:

$$I_N\ (f_1, f_2, \ldots f_N) = \sum_{K=2}^{N} KS_K\ F_K^N\ (\bar{V})$$

Subroutine FNCTN calculates one additive term $F_K^N\ [Q_M^K\ (U,V)]$ of the nonlinear current expression for given N, K, and M. Arguments transmitted to the subroutine are: N, the order of the current being calculated; K, the number of voltages in each transfer function product; M, the number of voltages from node V

in each product: U and V, the nodes from which the voltages are derived (relative to the base node of the nonlinear device)· and FREQ, the frequency combination vector. The calculated term is returned to the calling program in XX.

Subroutine FNCTN uses subroutine QFN to determine the node pattern returned in the integer vector Q, subroutine FUN to determine the orders of the transfer functions returned in the vector FNK, and subroutine FRPRM to produce the frequency permutations returned in the integer vector FREQC. Subroutine CROSS is called to access the voltages in the modified transfer function table and to form the voltage product for a single frequency permutation.

The algorithm used by subroutine FNCTN to calculate the term $F_K^N [Q_M^K (U, V)]$ is summarized as follows:

STEP 1·

a) Clear expression to XX=0.

b) Initialize FUN algorithm by ISWT=-1

c) Call subroutine QFN to generate $Q_M^K (U, V)$· return node pattern in Q vector.

STEP 2·

a) Initialize FRPRM algorithm by ISWF=-1

b) Clear frequency permutation count CNT=Ø.

c) Initialize symmetric product YY=Ø.

d) Call subroutine FUN to generate orders of transfer functions for one term of F_K^N ; return orders in FNK vector.

STEP 3:

a) Increment permutation count CNT=CNT+1.

b) Call subroutine FRPRM to create a frequency permutation; return permutation in FREQC vector.

c) Call subroutine CROSS to calculate product for this frequency permutation and add product to YY

d) If more permutations (ISWF=Ø), go to Step 3a.

e) If permutations complete (ISWF=1), go to Step 4.

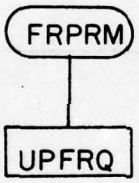
STEP 4:

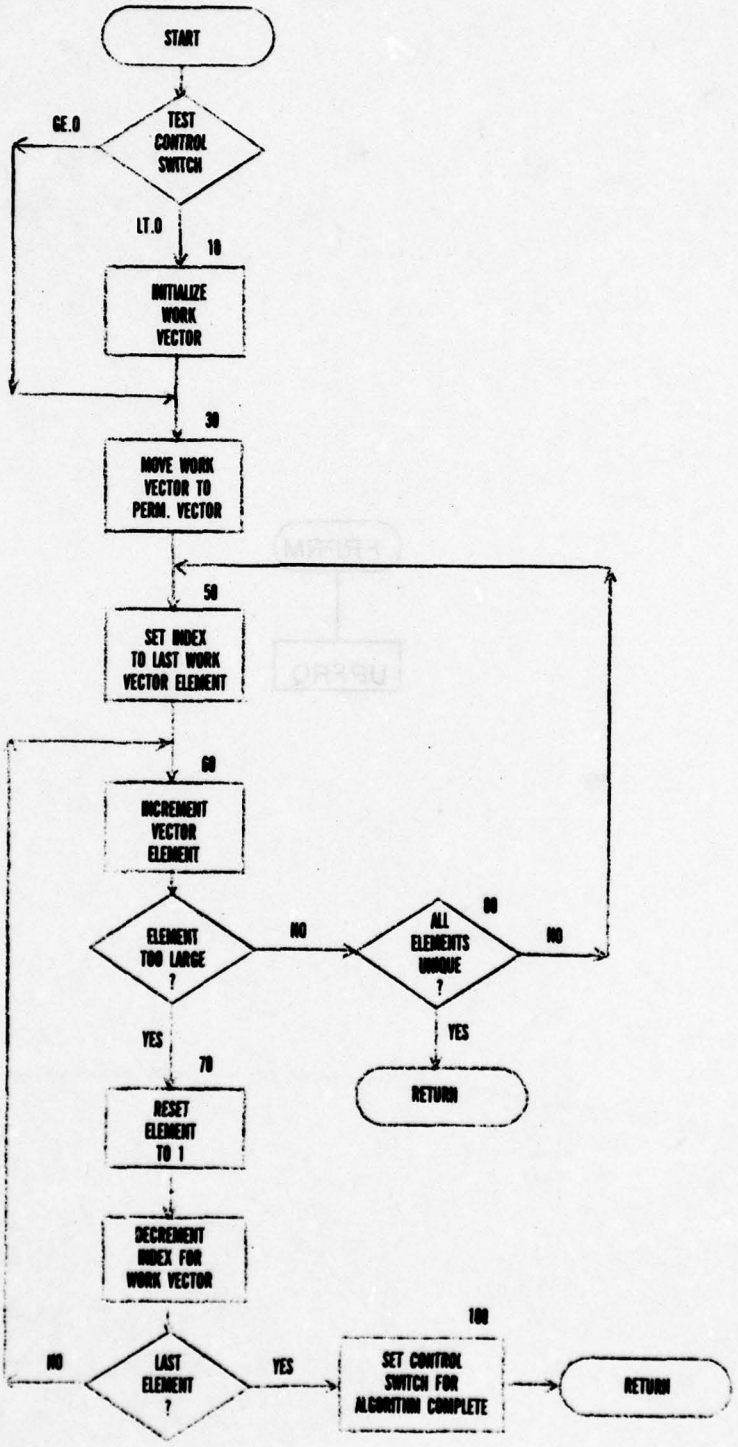
a) Calculate symmetric product and add to expression

XX=XX+YY/CNT

b) If more F_K^N terms (ISWF=Ø), go to Step 2b.

c) If F_K^N complete (ISWF=1), exit with calculated expression in XX.


FRPRM
UPFRQ


START
TEST CONTROL SWITCH
GE.0
LT.0
10
INITIALIZE WORK VECTOR
30
MOVE WORK VECTOR TO PERM. VECTOR
50
SET INDEX TO LAST WORK VECTOR ELEMENT
60
INCREMENT VECTOR ELEMENT
ELEMENT TOO LARGE ?
NO
80
ALL ELEMENTS UNIQUE ?
NO
YES
YES
70
RESET ELEMENT TO 1
RETURN
DECREMENT INDEX FOR WORK VECTOR
100
LAST ELEMENT ?
NO
YES
SET CONTROL SWITCH FOR ALGORITHM COMPLETE
RETURN

NAME: FRPRM

TYPE: SUBROUTINE

GENERAL PURPOSE:

Generates a frequency permutation for use by subroutine FNCTN

VARIABLES:

FREQ = The integers 1 thru 10

FREQC = Permuted frequency combination vector

ISW = Control switch: -1 = Initialize algorithm

0 = Continue algorithm

1 = Algorithm complete

N = Number of frequencies to be permuted

SEQ = Work vector

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

CURPT FNCTN

DESCRIPTION:

Subroutine FRPPM generates frequency permutations for use by subroutine FNCTN. Arguments transmitted to the subroutine are:

N, the number of frequencies to be permuted; FREQ, the sequential frequency numbers to be permuted; and ISW, a control variable described below. Arguments returned to the calling program are: FREQC, the permuted frequency vector and ISW, the updated control variable.

In order to generate every possible permutation of N frequency numbers, subroutine FRPRM must be invoked N! times. These iterative calls to subroutine FPPRM are controlled by the integer variable ISW, which is initialized to -1 by the calling program, tested and updated in FRPRM, then retested in the calling program. ISW=-1 initializes the algorithm of subroutine FRPRM by creating the first permutation of order N in the work vector SEQ. When ISW=0, subroutine FPPRM transfers the permutation from the work vector to FREQC and then generates the next permutation in SEQ, saving it in the labelled common area PRMFL. ISW is set to 1 when the last permutation has been generated.

The frequency permutations generated by subroutine FRPRM are stored in the first N elements of the integer vector SEQ. Each element of the vector is numerically equal to the sequential frequency number it represents. The largest frequency that any vector element can contain is N. Subroutine FRPRM ignores permutations in which a frequency number appears more than once.

Frequency permutations are generated recursively by subroutine FPPRM. The first permutation of N frequencies is defined to be the ordered set of integers (1, 2,...N) which is generated in the work vector by

SEQ(I)=I for I=1, 2,...N

Each successive permutation of order N is derived from the previous SEQ vector according to the following algorithm:

STEP 1 Initialize Algorithm at SEQ(N)

a) Set SEQ subscript to J=N.

STEP 2 Update and Test SEQ(J)

a) Define SEQ(J)=SEQ(J)+1

b) If (SEQ(J).GT.N), this value too large for SEQ(J). Go to Step 3.

c) If (SEQ(J).LE.N), this value O.K. for SEQ(J). Go to Step 4.

STEP 3 Redefine SEQ(J), Update and Test J

a) Set SEQ(J)=1

b) Decrement SEQ subscript J=J-1

c) If (J.GT.0), go to Step 2. Otherwise algorithm complete.

Step 4: Test SEQ for Uniqueness

a) If SEQ(I)=SEQ(J) for any I and J (I≠J), go to Step 1.

b) Otherwise algorithm complete.

A tabulation of all frequency permutations for every order up to N=6 (the largest order allowed in NCAP) is presented below:

n = 2

NO.	FREQC 1	2	3	4	5	6
1	1	2	0	0	0	0
2	2	1	0	0	0	0

n = 3

NO.	FREQC 1	2	3	4	5	6
1	1	2	3	0	0	0
2	1	3	2	0	0	0
3	2	1	3	0	0	0
4	2	3	1	0	0	0
5	3	1	2	0	0	0
6	3	2	1	0	0	0

n = 4

NO.	FREQC 1	2	3	4	5	6
1	1	2	3	4	0	0
2	1	2	4	3	0	0
3	1	3	2	4	0	0
4	1	3	4	2	0	0
5	1	4	2	3	0	0
6	1	4	3	2	0	0
7	2	1	3	4	0	0
8	2	1	4	3	0	0
9	2	3	1	4	0	0
10	2	3	4	1	0	0
11	2	4	1	3	0	0
12	2	4	3	1	0	0
13	3	1	2	4	0	0
14	3	1	4	2	0	0
15	3	2	1	4	0	0
16	3	2	4	1	0	0
17	3	4	1	2	0	0
18	3	4	2	1	0	0
19	4	1	2	3	0	0
20	4	1	3	2	0	0
21	4	2	1	3	0	0
22	4	2	3	1	0	0
23	4	3	1	2	0	0
24	4	3	2	1	0	0

n = 5

NO.	FREQC 1	2	3	4	5	6
1	1	2	3	4	5	0
2	1	2	3	5	4	0
3	1	2	4	3	5	0
4	1	2	4	5	3	0
5	1	2	5	3	4	0
6	1	2	5	4	3	0
7	1	3	2	4	5	0
8	1	3	2	5	4	0
9	1	3	4	2	5	0
10	1	3	4	5	2	0
11	1	3	5	2	4	0
12	1	3	5	4	2	0
13	1	4	2	3	5	0
14	1	4	2	5	3	0
15	1	4	3	2	5	0
16	1	4	3	5	2	0
17	1	4	5	2	3	0
18	1	4	5	3	2	0
19	1	5	2	3	4	0
20	1	5	2	4	3	0
21	1	5	3	2	4	0
22	1	5	3	4	2	0
23	1	5	4	2	3	0
24	1	5	4	3	2	0
25	2	1	3	4	5	0
26	2	1	3	5	4	0
27	2	1	4	3	5	0
28	2	1	4	5	3	0
29	2	1	5	3	4	0
30	2	1	5	4	3	0
31	2	3	1	4	5	0
32	2	3	1	5	4	0
33	2	3	4	1	5	0
34	2	3	4	5	1	0
35	2	3	5	1	4	0
36	2	3	5	4	1	0
37	2	4	1	3	5	0
38	2	4	1	5	3	0
39	2	4	3	1	5	0
40	2	4	3	5	1	0
41	2	4	5	1	3	0
42	2	4	5	3	1	0
43	2	5	1	3	4	0
44	2	5	1	4	3	0
45	2	5	3	1	4	0
46	2	5	3	4	1	0
47	2	5	4	1	3	0
48	2	5	4	3	1	0
49	3	1	2	4	5	0
50	3	1	2	5	4	0
51	3	1	4	2	5	0
52	3	1	4	5	2	0
53	3	1	5	2	4	0
54	3	1	5	4	2	0
55	3	2	1	4	5	0
56	3	2	1	5	4	0
57	3	2	4	1	5	0
58	3	2	4	5	1	0
59	3	2	5	1	4	0
60	3	2	5	4	1	0

NO.	FREQC 1	2	3	4	5	6
61	3	4	1	2	5	0
62	3	4	1	5	2	0
63	3	4	2	1	5	0
64	3	4	2	5	1	0
65	3	4	5	1	2	0
66	3	4	5	2	1	0
67	3	5	1	2	4	0
68	3	5	1	4	2	0
69	3	5	2	1	4	0
70	3	5	2	4	1	0
71	3	5	4	1	2	0
72	3	5	4	2	1	0
73	4	1	2	3	5	0
74	4	1	2	5	3	0
75	4	1	3	2	5	0
76	4	1	3	5	2	0
77	4	1	5	2	3	0
78	4	1	5	3	2	0
79	4	2	1	3	5	0
80	4	2	1	5	3	0
81	4	2	3	1	5	0
82	4	2	3	5	1	0
83	4	2	5	1	3	0
84	4	2	5	3	1	0
85	4	3	1	2	5	0
86	4	3	1	5	2	0
87	4	3	2	1	5	0
88	4	3	2	5	1	0
89	4	3	5	1	2	0
90	4	3	5	2	1	0
91	4	5	1	2	3	0
92	4	5	1	3	2	0
93	4	5	2	1	3	0
94	4	5	2	3	1	0
95	4	5	3	1	2	0
96	4	5	3	2	1	0
97	5	1	2	3	4	0
98	5	1	2	4	3	0
99	5	1	3	2	4	0
100	5	1	3	4	2	0
101	5	1	4	2	3	0
102	5	1	4	3	2	0
103	5	2	1	3	4	0
104	5	2	1	4	3	0
105	5	2	3	1	4	0
106	5	2	3	4	1	0
107	5	2	4	1	3	0
108	5	2	4	3	1	0
109	5	3	1	2	4	0
110	5	3	1	4	2	0
111	5	3	2	1	4	0
112	5	3	2	4	1	0
113	5	3	4	1	2	0
114	5	3	4	2	1	0
115	5	4	1	2	3	0
116	5	4	1	3	2	0
117	5	4	2	1	3	0
118	5	4	2	3	1	0
119	5	4	3	1	2	0
120	5	4	3	2	1	0

n = 6

NO.	FREQC 1	2	3	4	5	6
1	1	2	3	4	5	6
2	1	2	3	4	6	5
3	1	2	3	5	4	6
4	1	2	3	5	6	4
5	1	2	3	6	4	5
6	1	2	3	6	5	4
7	1	2	4	3	5	6
8	1	2	4	3	6	5
9	1	2	4	5	3	6
10	1	2	4	5	6	3
11	1	2	4	6	3	5
12	1	2	4	6	5	3
13	1	2	5	3	4	6
14	1	2	5	3	6	4
15	1	2	5	4	3	6
16	1	2	5	4	6	3
17	1	2	5	6	3	4
18	1	2	5	6	4	3
19	1	2	6	3	4	5
20	1	2	6	3	5	4
21	1	2	6	4	3	5
22	1	2	6	4	5	3
23	1	2	6	5	3	4
24	1	2	6	5	4	3
25	1	3	2	4	5	6
26	1	3	2	4	6	5
27	1	3	2	5	4	6
28	1	3	2	5	6	4
29	1	3	2	6	4	5
30	1	3	2	6	5	4
31	1	3	4	2	5	6
32	1	3	4	2	6	5
33	1	3	4	5	2	6
34	1	3	4	5	6	2
35	1	3	4	6	2	5
36	1	3	4	6	5	2
37	1	3	5	2	4	6
38	1	3	5	2	6	4
39	1	3	5	4	2	6
40	1	3	5	4	6	2
41	1	3	5	6	2	4
42	1	3	5	6	4	2
43	1	3	6	2	4	5
44	1	3	6	2	5	4
45	1	3	6	4	2	5
46	1	3	6	4	5	2
47	1	3	6	5	2	4
48	1	3	6	5	4	2
49	1	4	2	3	5	6
50	1	4	2	3	6	5
51	1	4	2	5	3	6
52	1	4	2	5	6	3
53	1	4	2	6	3	5
54	1	4	2	6	5	3
55	1	4	3	2	5	6
56	1	4	3	2	6	5
57	1	4	3	5	2	6
58	1	4	3	5	6	2
59	1	4	3	6	2	5
60	1	4	3	6	5	2

NO.	FREQC 1	2	3	4	5	6
61	1	4	5	2	3	6
62	1	4	5	2	6	3
63	1	4	5	3	2	6
64	1	4	5	3	6	2
65	1	4	5	6	2	3
66	1	4	5	6	3	2
67	1	4	6	2	3	5
68	1	4	6	2	5	3
69	1	4	6	3	2	5
70	1	4	6	3	5	2
71	1	4	6	5	2	3
72	1	4	6	5	3	2
73	1	5	2	3	4	6
74	1	5	2	3	6	4
75	1	5	2	4	3	6
76	1	5	2	4	6	3
77	1	5	2	6	3	4
78	1	5	2	6	4	3
79	1	5	3	2	4	6
80	1	5	3	2	6	4
81	1	5	3	4	2	6
82	1	5	3	4	6	2
83	1	5	3	6	2	4
84	1	5	3	6	4	2
85	1	5	4	2	3	6
86	1	5	4	2	6	3
87	1	5	4	3	2	6
88	1	5	4	3	6	2
89	1	5	4	6	2	3
90	1	5	4	6	3	2
91	1	5	6	2	3	4
92	1	5	6	2	4	3
93	1	5	6	3	2	4
94	1	5	6	3	4	2
95	1	5	6	4	2	3
96	1	5	6	4	3	2
97	1	6	2	3	4	5
98	1	6	2	3	5	4
99	1	6	2	4	3	5
100	1	6	2	4	5	3
101	1	6	2	5	3	4
102	1	6	2	5	4	3
103	1	6	3	2	4	5
104	1	6	3	2	5	4
105	1	6	3	4	2	5
106	1	6	3	4	5	2
107	1	6	3	5	2	4
108	1	6	3	5	4	2
109	1	6	4	2	3	5
110	1	6	4	2	5	3
111	1	6	4	3	2	5
112	1	6	4	3	5	2
113	1	6	4	5	2	3
114	1	6	4	5	3	2
115	1	6	5	2	3	4
116	1	6	5	2	4	3
117	1	6	5	3	2	4
118	1	6	5	3	4	2
119	1	6	5	4	2	3
120	1	6	5	4	3	2

n = 6 (continued)

NO.	FREQC 1	2	3	4	5	6
121	2	1	3	4	5	6
122	2	1	3	4	6	5
123	2	1	3	5	4	6
124	2	1	3	5	6	4
125	2	1	3	6	4	5
126	2	1	3	6	5	4
127	2	1	4	3	5	6
128	2	1	4	3	6	5
129	2	1	4	5	3	6
130	2	1	4	5	6	3
131	2	1	4	6	3	5
132	2	1	4	6	5	3
133	2	1	5	3	4	6
134	2	1	5	3	6	4
135	2	1	5	4	3	6
136	2	1	5	4	6	3
137	2	1	5	6	3	4
138	2	1	5	6	4	3
139	2	1	6	3	4	5
140	2	1	6	3	5	4
141	2	1	6	4	3	5
142	2	1	6	4	5	3
143	2	1	6	5	3	4
144	2	1	6	5	4	3
145	2	3	1	4	5	6
146	2	3	1	4	6	5
147	2	3	1	5	4	6
148	2	3	1	5	6	4
149	2	3	1	6	4	5
150	2	3	1	6	5	4
151	2	3	4	1	5	6
152	2	3	4	1	6	5
153	2	3	4	5	1	6
154	2	3	4	5	6	1
155	2	3	4	6	1	5
156	2	3	4	6	5	1
157	2	3	5	1	4	6
158	2	3	5	1	6	4
159	2	3	5	4	1	6
160	2	3	5	4	6	1
161	2	3	5	6	1	4
162	2	3	5	6	4	1
163	2	3	6	1	4	5
164	2	3	6	1	5	4
165	2	3	6	4	1	5
166	2	3	6	4	5	1
167	2	3	6	5	1	4
168	2	3	6	5	4	1
169	2	4	1	3	5	6
170	2	4	1	3	6	5
171	2	4	1	5	3	6
172	2	4	1	5	6	3
173	2	4	1	6	3	5
174	2	4	1	6	5	3
175	2	4	3	1	5	6
176	2	4	3	1	6	5
177	2	4	3	5	1	6
178	2	4	3	5	6	1
179	2	4	3	6	1	5
180	2	4	3	6	5	1

NO.	FREQC 1	2	3	4	5	6
181	2	4	5	1	3	6
182	2	4	5	1	6	3
183	2	4	5	3	1	6
184	2	4	5	3	6	1
185	2	4	5	6	1	3
186	2	4	5	6	3	1
187	2	4	6	1	3	5
188	2	4	6	1	5	3
189	2	4	6	3	1	5
190	2	4	6	3	5	1
191	2	4	6	5	1	3
192	2	4	6	5	3	1
193	2	5	1	3	4	6
194	2	5	1	3	6	4
195	2	5	1	4	3	6
196	2	5	1	4	6	3
197	2	5	1	6	3	4
198	2	5	1	6	4	3
199	2	5	3	1	4	6
200	2	5	3	1	6	4
201	2	5	3	4	1	6
202	2	5	3	4	6	1
203	2	5	3	6	1	4
204	2	5	3	6	4	1
205	2	5	4	1	3	6
206	2	5	4	1	6	3
207	2	5	4	3	1	6
208	2	5	4	3	6	1
209	2	5	4	6	1	3
210	2	5	4	6	3	1
211	2	5	6	1	3	4
212	2	5	6	1	4	3
213	2	5	6	3	1	4
214	2	5	6	3	4	1
215	2	5	6	4	1	3
216	2	5	6	4	3	1
217	2	6	1	3	4	5
218	2	6	1	3	5	4
219	2	6	1	4	3	5
220	2	6	1	4	5	3
221	2	6	1	5	3	4
222	2	6	1	5	4	3
223	2	6	3	1	4	5
224	2	6	3	1	5	4
225	2	6	3	4	1	5
226	2	6	3	4	5	1
227	2	6	3	5	1	4
228	2	6	3	5	4	1
229	2	6	4	1	3	5
230	2	6	4	1	5	3
231	2	6	4	3	1	5
232	2	6	4	3	5	1
233	2	6	4	5	1	3
234	2	6	4	5	3	1
235	2	6	5	1	3	4
236	2	6	5	1	4	3
237	2	6	5	3	1	4
238	2	6	5	3	4	1
239	2	6	5	4	1	3
240	2	6	5	4	3	1

n = 6 (continued)

NO.	FREQC 1	2	3	4	5	6
241	3	1	2	4	5	6
242	3	1	2	4	6	5
243	3	1	2	5	4	6
244	3	1	2	5	6	4
245	3	1	2	6	4	5
246	3	1	2	6	5	4
247	3	1	4	2	5	6
248	3	1	4	2	6	5
249	3	1	4	5	2	6
250	3	1	4	5	6	2
251	3	1	4	6	2	5
252	3	1	4	6	5	2
253	3	1	5	2	4	6
254	3	1	5	2	6	4
255	3	1	5	4	2	6
256	3	1	5	4	6	2
257	3	1	5	6	2	4
258	3	1	5	6	4	2
259	3	1	6	2	4	5
260	3	1	6	2	5	4
261	3	1	6	4	2	5
262	3	1	6	4	5	2
263	3	1	6	5	2	4
264	3	1	6	5	4	2
265	3	2	1	4	5	6
266	3	2	1	4	6	5
267	3	2	1	5	4	6
268	3	2	1	5	6	4
269	3	2	1	6	4	5
270	3	2	1	6	5	4
271	3	2	4	1	5	6
272	3	2	4	1	6	5
273	3	2	4	5	1	6
274	3	2	4	5	6	1
275	3	2	4	6	1	5
276	3	2	4	6	5	1
277	3	2	5	1	4	6
278	3	2	5	1	6	4
279	3	2	5	4	1	6
280	3	2	5	4	6	1
281	3	2	5	6	1	4
282	3	2	5	6	4	1
283	3	2	6	1	4	5
284	3	2	6	1	5	4
285	3	2	6	4	1	5
286	3	2	6	4	5	1
287	3	2	6	5	1	4
288	3	2	6	5	4	1
289	3	4	1	2	5	6
290	3	4	1	2	6	5
291	3	4	1	5	2	6
292	3	4	1	5	6	2
293	3	4	1	6	2	5
294	3	4	1	6	5	2
295	3	4	2	1	5	6
296	3	4	2	1	6	5
297	3	4	2	5	1	6
298	3	4	2	5	6	1
299	3	4	2	6	1	5
300	3	4	2	6	5	1

NO.	FREQC 1	2	3	4	5	6
301	3	4	5	1	2	6
302	3	4	5	1	6	2
303	3	4	5	2	1	6
304	3	4	5	2	6	1
305	3	4	5	6	1	2
306	3	4	5	6	2	1
307	3	4	6	1	2	5
308	3	4	6	1	5	2
309	3	4	6	2	1	5
310	3	4	6	2	5	1
311	3	4	6	5	1	2
312	3	4	6	5	2	1
313	3	5	1	2	4	6
314	3	5	1	2	6	4
315	3	5	1	4	2	6
316	3	5	1	4	6	2
317	3	5	1	6	2	4
318	3	5	1	6	4	2
319	3	5	2	1	4	6
320	3	5	2	1	6	4
321	3	5	2	4	1	6
322	3	5	2	4	6	1
323	3	5	2	6	1	4
324	3	5	2	6	4	1
325	3	5	4	1	2	6
326	3	5	4	1	6	2
327	3	5	4	2	1	6
328	3	5	4	2	6	1
329	3	5	4	6	1	2
330	3	5	4	6	2	1
331	3	5	6	1	2	4
332	3	5	6	1	4	2
333	3	5	6	2	1	4
334	3	5	6	2	4	1
335	3	5	6	4	1	2
336	3	5	6	4	2	1
337	3	6	1	2	4	5
338	3	6	1	2	5	4
339	3	6	1	4	2	5
340	3	6	1	4	5	2
341	3	6	1	5	2	4
342	3	6	1	5	4	2
343	3	6	2	1	4	5
344	3	6	2	1	5	4
345	3	6	2	4	1	5
346	3	6	2	4	5	1
347	3	6	2	5	1	4
348	3	6	2	5	4	1
349	3	6	4	1	2	5
350	3	6	4	1	5	2
351	3	6	4	2	1	5
352	3	6	4	2	5	1
353	3	6	4	5	1	2
354	3	6	4	5	2	1
355	3	6	5	1	2	4
356	3	6	5	1	4	2
357	3	6	5	2	1	4
358	3	6	5	2	4	1
359	3	6	5	4	1	2
360	3	6	5	4	2	1

n = 6 (continued)

NO.	FREQC 1	2	3	4	5	6
361	4	1	2	3	5	6
362	4	1	2	3	6	5
363	4	1	2	5	3	6
364	4	1	2	5	6	3
365	4	1	2	6	3	5
366	4	1	2	6	5	3
367	4	1	3	2	5	6
368	4	1	3	2	6	5
369	4	1	3	5	2	6
370	4	1	3	5	6	2
371	4	1	3	6	2	5
372	4	1	3	6	5	2
373	4	1	5	2	3	6
374	4	1	5	2	6	3
375	4	1	5	3	2	6
376	4	1	5	3	6	2
377	4	1	5	6	2	3
378	4	1	5	6	3	2
379	4	1	6	2	3	5
380	4	1	6	2	5	3
381	4	1	6	3	2	5
382	4	1	6	3	5	2
383	4	1	6	5	2	3
384	4	1	6	5	3	2
385	4	2	1	3	5	6
386	4	2	1	3	6	5
387	4	2	1	5	3	6
388	4	2	1	5	6	3
389	4	2	1	6	3	5
390	4	2	1	6	5	3
391	4	2	3	1	5	6
392	4	2	3	1	6	5
393	4	2	3	5	1	6
394	4	2	3	5	6	1
395	4	2	3	6	1	5
396	4	2	3	6	5	1
397	4	2	5	1	3	6
398	4	2	5	1	6	3
399	4	2	5	3	1	6
400	4	2	5	3	6	1
401	4	2	5	6	1	3
402	4	2	5	6	3	1
403	4	2	6	1	3	5
404	4	2	6	1	5	3
405	4	2	6	3	1	5
406	4	2	6	3	5	1
407	4	2	6	5	1	3
408	4	2	6	5	3	1
409	4	3	1	2	5	6
410	4	3	1	2	6	5
411	4	3	1	5	2	6
412	4	3	1	5	6	2
413	4	3	1	6	2	5
414	4	3	1	6	5	2
415	4	3	2	1	5	6
416	4	3	2	1	6	5
417	4	3	2	5	1	6
418	4	3	2	5	6	1
419	4	3	2	6	1	5
420	4	3	2	6	5	1

NO.	FREQC 1	2	3	4	5	6
421	4	3	5	1	2	6
422	4	3	5	1	6	2
423	4	3	5	2	1	6
424	4	3	5	2	6	1
425	4	3	5	6	1	2
426	4	3	5	6	2	1
427	4	3	6	1	2	5
428	4	3	6	1	5	2
429	4	3	6	2	1	5
430	4	3	6	2	5	1
431	4	3	6	5	1	2
432	4	3	6	5	2	1
433	4	5	1	2	3	6
434	4	5	1	2	6	3
435	4	5	1	3	2	6
436	4	5	1	3	6	2
437	4	5	1	6	2	3
438	4	5	1	6	3	2
439	4	5	2	1	3	6
440	4	5	2	1	6	3
441	4	5	2	3	1	6
442	4	5	2	3	6	1
443	4	5	2	6	1	3
444	4	5	2	6	3	1
445	4	5	3	1	2	6
446	4	5	3	1	6	2
447	4	5	3	2	1	6
448	4	5	3	2	6	1
449	4	5	3	6	1	2
450	4	5	3	6	2	1
451	4	5	6	1	2	3
452	4	5	6	1	3	2
453	4	5	6	2	1	3
454	4	5	6	2	3	1
455	4	5	6	3	1	2
456	4	5	6	3	2	1
457	4	6	1	2	3	5
458	4	6	1	2	5	3
459	4	6	1	3	2	5
460	4	6	1	3	5	2
461	4	6	1	5	2	3
462	4	6	1	5	3	2
463	4	6	2	1	3	5
464	4	6	2	1	5	3
465	4	6	2	3	1	5
466	4	6	2	3	5	1
467	4	6	2	5	1	3
468	4	6	2	5	3	1
469	4	6	3	1	2	5
470	4	6	3	1	5	2
471	4	6	3	2	1	5
472	4	6	3	2	5	1
473	4	6	3	5	1	2
474	4	6	3	5	2	1
475	4	6	5	1	2	3
476	4	6	5	1	3	2
477	4	6	5	2	1	3
478	4	6	5	2	3	1
479	4	6	5	3	1	2
480	4	6	5	3	2	1

n = 6 (continued)

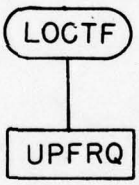
NO.	FREQC 1	2	3	4	5	6
481	5	1	2	3	4	6
482	5	1	2	3	6	4
483	5	1	2	4	3	6
484	5	1	2	4	6	3
485	5	1	2	6	3	4
486	5	1	2	6	4	3
487	5	1	3	2	4	6
488	5	1	3	2	6	4
489	5	1	3	4	2	6
490	5	1	3	4	6	2
491	5	1	3	6	2	4
492	5	1	3	6	4	2
493	5	1	4	2	3	6
494	5	1	4	2	6	3
495	5	1	4	3	2	6
496	5	1	4	3	6	2
497	5	1	4	6	2	3
498	5	1	4	6	3	2
499	5	1	6	2	3	4
500	5	1	6	2	4	3
501	5	1	6	3	2	4
502	5	1	6	3	4	2
503	5	1	6	4	2	3
504	5	1	6	4	3	2
505	5	2	1	3	4	6
506	5	2	1	3	6	4
507	5	2	1	4	3	6
508	5	2	1	4	6	3
509	5	2	1	6	3	4
510	5	2	1	6	4	3
511	5	2	3	1	4	6
512	5	2	3	1	6	4
513	5	2	3	4	1	6
514	5	2	3	4	6	1
515	5	2	3	6	1	4
516	5	2	3	6	4	1
517	5	2	4	1	3	6
518	5	2	4	1	6	3
519	5	2	4	3	1	6
520	5	2	4	3	6	1
521	5	2	4	6	1	3
522	5	2	4	6	3	1
523	5	2	6	1	3	4
524	5	2	6	1	4	3
525	5	2	6	3	1	4
526	5	2	6	3	4	1
527	5	2	6	4	1	3
528	5	2	6	4	3	1
529	5	3	1	2	4	6
530	5	3	1	2	6	4
531	5	3	1	4	2	6
532	5	3	1	4	6	2
533	5	3	1	6	2	4
534	5	3	1	6	4	2
535	5	3	2	1	4	6
536	5	3	2	1	6	4
537	5	3	2	4	1	6
538	5	3	2	4	6	1
539	5	3	2	6	1	4
540	5	3	2	6	4	1

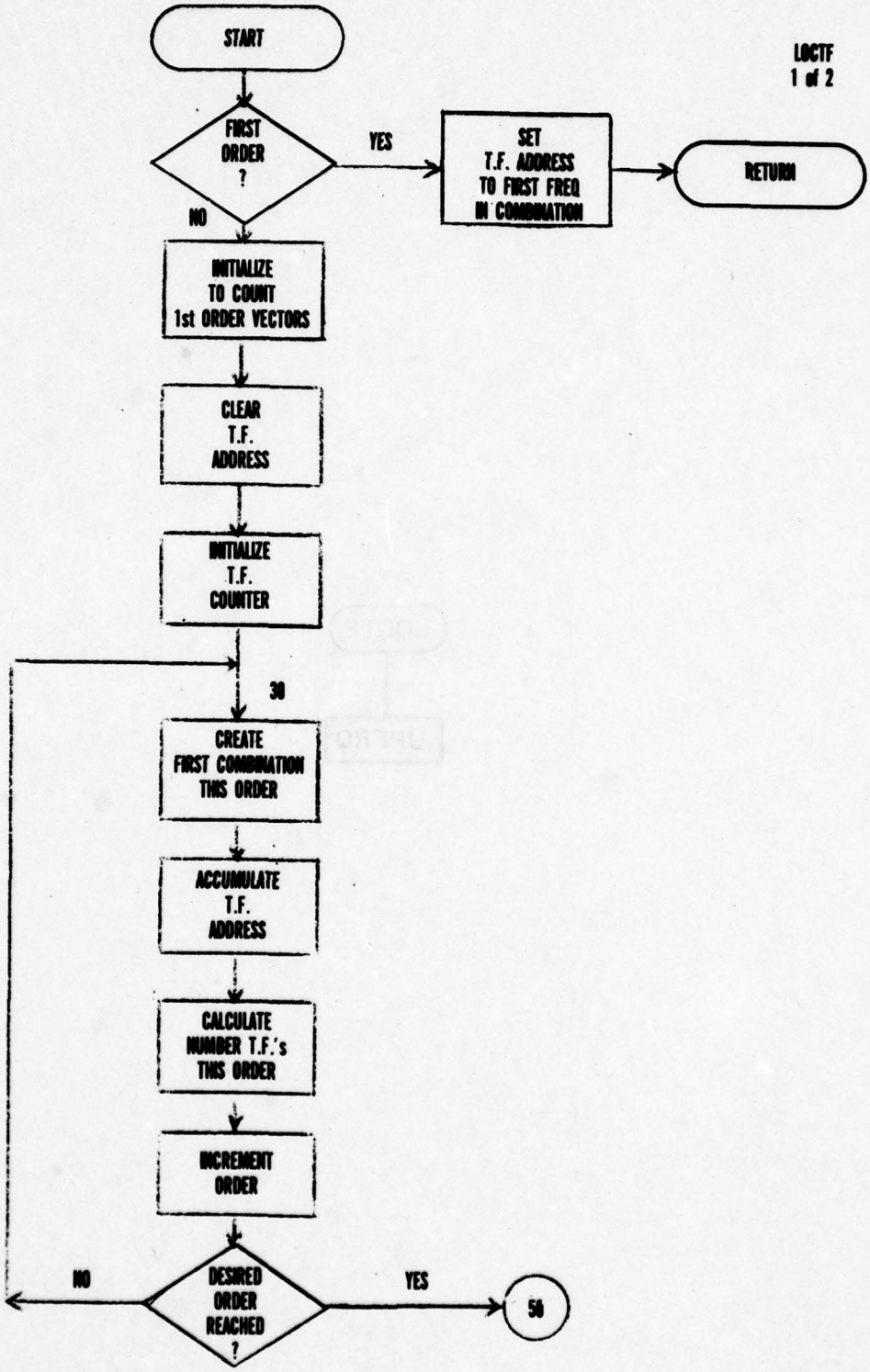
NO.	FREQC 1	2	3	4	5	6
541	5	3	4	1	2	6
542	5	3	4	1	6	2
543	5	3	4	2	1	6
544	5	3	4	2	6	1
545	5	3	4	6	1	2
546	5	3	4	6	2	1
547	5	3	6	1	2	4
548	5	3	6	1	4	2
549	5	3	6	2	1	4
550	5	3	6	2	4	1
551	5	3	6	4	1	2
552	5	3	6	4	2	1
553	5	4	1	2	3	6
554	5	4	1	2	6	3
555	5	4	1	3	2	6
556	5	4	1	3	6	2
557	5	4	1	6	2	3
558	5	4	1	6	3	2
559	5	4	2	1	3	6
560	5	4	2	1	6	3
561	5	4	2	3	1	6
562	5	4	2	3	6	1
563	5	4	2	6	1	3
564	5	4	2	6	3	1
565	5	4	3	1	2	6
566	5	4	3	1	6	2
567	5	4	3	2	1	6
568	5	4	3	2	6	1
569	5	4	3	6	1	2
570	5	4	3	6	2	1
571	5	4	6	1	2	3
572	5	4	6	1	3	2
573	5	4	6	2	1	3
574	5	4	6	2	3	1
575	5	4	6	3	1	2
576	5	4	6	3	2	1
577	5	6	1	2	3	4
578	5	6	1	2	4	3
579	5	6	1	3	2	4
580	5	6	1	3	4	2
581	5	6	1	4	2	3
582	5	6	1	4	3	2
583	5	6	2	1	3	4
584	5	6	2	1	4	3
585	5	6	2	3	1	4
586	5	6	2	3	4	1
587	5	6	2	4	1	3
588	5	6	2	4	3	1
589	5	6	3	1	2	4
590	5	6	3	1	4	2
591	5	6	3	2	1	4
592	5	6	3	2	4	1
593	5	6	3	4	1	2
594	5	6	3	4	2	1
595	5	6	4	1	2	3
596	5	6	4	1	3	2
597	5	6	4	2	1	3
598	5	6	4	2	3	1
599	5	6	4	3	1	2
600	5	6	4	3	2	1

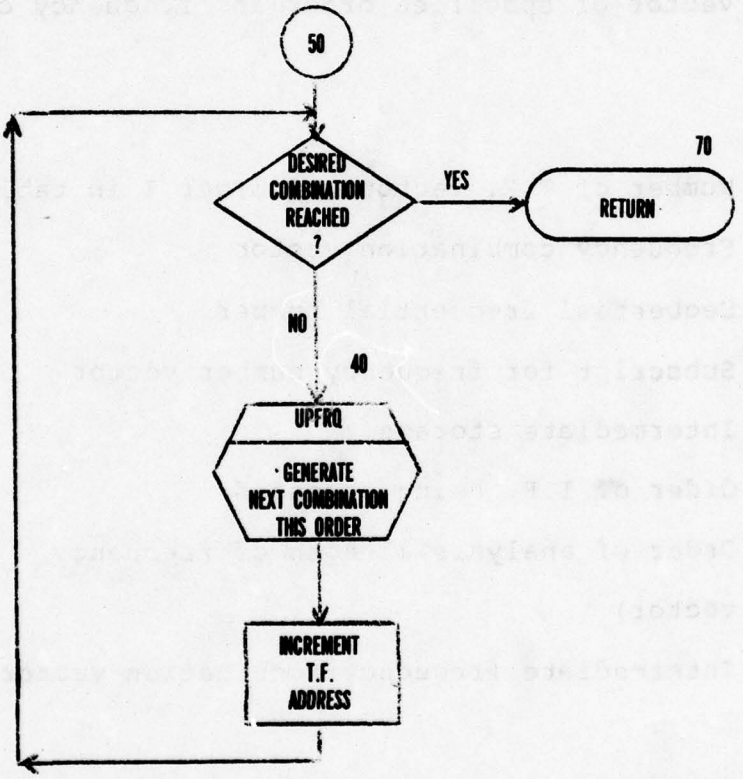
n = 6 (continued)

NO.	FREQC 1	2	3	4	5	6
601	6	1	2	3	4	5
602	6	1	2	3	5	4
603	6	1	2	4	3	5
604	6	1	2	4	5	3
605	6	1	2	5	3	4
606	6	1	2	5	4	3
607	6	1	3	2	4	5
608	6	1	3	2	5	4
609	6	1	3	4	2	5
610	6	1	3	4	5	2
611	6	1	3	5	2	4
612	6	1	3	5	4	2
613	6	1	4	2	3	5
614	6	1	4	2	5	3
615	6	1	4	3	2	5
616	6	1	4	3	5	2
617	6	1	4	5	2	3
618	6	1	4	5	3	2
619	6	1	5	2	3	4
620	6	1	5	2	4	3
621	6	1	5	3	2	4
622	6	1	5	3	4	2
623	6	1	5	4	2	3
624	6	1	5	4	3	2
625	6	2	1	3	4	5
626	6	2	1	3	5	4
627	6	2	1	4	3	5
628	6	2	1	4	5	3
629	6	2	1	5	3	4
630	6	2	1	5	4	3
631	6	2	3	1	4	5
632	6	2	3	1	5	4
633	6	2	3	4	1	5
634	6	2	3	4	5	1
635	6	2	3	5	1	4
636	6	2	3	5	4	1
637	6	2	4	1	3	5
638	6	2	4	1	5	3
639	6	2	4	3	1	5
640	6	2	4	3	5	1
641	6	2	4	5	1	3
642	6	2	4	5	3	1
643	6	2	5	1	3	4
644	6	2	5	1	4	3
645	6	2	5	3	1	4
646	6	2	5	3	4	1
647	6	2	5	4	1	3
648	6	2	5	4	3	1
649	6	3	1	2	4	5
650	6	3	1	2	5	4
651	6	3	1	4	2	5
652	6	3	1	4	5	2
653	6	3	1	5	2	4
654	6	3	1	5	4	2
655	6	3	2	1	4	5
656	6	3	2	1	5	4
657	6	3	2	4	1	5
658	6	3	2	4	5	1
659	6	3	2	5	1	4
660	6	3	2	5	4	1

NO.	FREQC 1	2	3	4	5	6
661	6	3	4	1	2	5
662	6	3	4	1	5	2
663	6	3	4	2	1	5
664	6	3	4	2	5	1
665	6	3	4	5	1	2
666	6	3	4	5	2	1
667	6	3	5	1	2	4
668	6	3	5	1	4	2
669	6	3	5	2	1	4
670	6	3	5	2	4	1
671	6	3	5	4	1	2
672	6	3	5	4	2	1
673	6	4	1	2	3	5
674	6	4	1	2	5	3
675	6	4	1	3	2	5
676	6	4	1	3	5	2
677	6	4	1	5	2	3
678	6	4	1	5	3	2
679	6	4	2	1	3	5
680	6	4	2	1	5	3
681	6	4	2	3	1	5
682	6	4	2	3	5	1
683	6	4	2	5	1	3
684	6	4	2	5	3	1
685	6	4	3	1	2	5
686	6	4	3	1	5	2
687	6	4	3	2	1	5
688	6	4	3	2	5	1
689	6	4	3	5	1	2
690	6	4	3	5	2	1
691	6	4	5	1	2	3
692	6	4	5	1	3	2
693	6	4	5	2	1	3
694	6	4	5	2	3	1
695	6	4	5	3	1	2
696	6	4	5	3	2	1
697	6	5	1	2	3	4
698	6	5	1	2	4	3
699	6	5	1	3	2	4
700	6	5	1	3	4	2
701	6	5	1	4	2	3
702	6	5	1	4	3	2
703	6	5	2	1	3	4
704	6	5	2	1	4	3
705	6	5	2	3	1	4
706	6	5	2	3	4	1
707	6	5	2	4	1	3
708	6	5	2	4	3	1
709	6	5	3	1	2	4
710	6	5	3	1	4	2
711	6	5	3	2	1	4
712	6	5	3	2	4	1
713	6	5	3	4	1	2
714	6	5	3	4	2	1
715	6	5	4	1	2	3
716	6	5	4	1	3	2
717	6	5	4	2	1	3
718	6	5	4	2	3	1
719	6	5	4	3	1	2
720	6	5	4	3	2	1


LOCTF
UPFRQ


START
LOCTF
1 of 2
FIRST ORDER ?
YES
SET T.F. ADDRESS TO FIRST FREQ IN COMBINATION
RETURN
NO
INITIALIZE TO COUNT 1st ORDER VECTORS
CLEAR T.F. ADDRESS
INITIALIZE T.F. COUNTER
30
CREATE FIRST COMBINATION THIS ORDER
ACCUMULATE T.F. ADDRESS
CALCULATE NUMBER T.F.'s THIS ORDER
INCREMENT ORDER
NO
DESIRED ORDER REACHED ?
YES
50


50
DESIRED COMBINATION REACHED ?
YES
70
RETURN
NO
40
UPFRQ
GENERATE NEXT COMBINATION THIS ORDER
INCREMENT T.F. ADDRESS

NAME: LOCTF

TYPE: FUNCTION

GENERAL PURPOSE:

Calculates file 24 record number of a modified transfer function vector of specified order and frequency combination

VARIABLES:

C = Number of T.F. vectors of order I in table

FRQS = Frequency combination vector

I = Sequential frequential number

J = Subscript for frequency number vector

L = Intermediate storage

M = Order of T.F. being accessed

N = Order of analysis (length of frequency vector)

SEQ = Intermediate frequency combination vector

SUBROUTINES CALLED:

UPFRQ

CALLING PROGRAMS:

CROSS

DESCRIPTION·

Function LOCTF calculates the file 24 address of the modified transfer function vector of a specified order and frequency combination for use in computing nonlinear currents in Phase 3. Arguments transmitted to the function are: N, the order of the nonlinear current being computed: M, the order of the transfer function vector being accessed; and FRQS, the frequency combination of the transfer function vector being accessed. The calculated address is returned to the calling program in LOCTF.

During the calculation of Nth order nonlinear currents, the transfer function table contains $2^n - 2$ lower-order transfer function vectors. These vectors are stored according to increasing frequency combinations in the order they were created in Phase 3. The address of a particular vector is a function of its order and frequency combination, and can be calculated by finding the address of the first vector of like order and adding a displacement factor based on the frequency combination of the desired vector.

There are exactly $\binom{N}{I}$ transfer function vectors of each order I=1, 2,...M in the table. Therefore the first vector of order M is located at·

$$\text{LOCTF} = 1 + \sum_{I=1}^{M-1} \binom{N}{I}$$

The frequency combination corresponding to this vector is the ordered set of sequential frequency numbers (1,2,...M). This frequency combination is created in the first M elements of the integer vector SEQ. Each element of SEQ is numerically equal to the frequency number it represents and the frequencies are stored in the vector so that they decrease numerically as the subscript of SEQ increases.

The displacement factor is added to LOCTF by successively updating the frequency combination while at the same time incrementing LOCTF. Subroutine UPFRQ updates the frequency combination. When the calculated frequency combination matches FRQS, the frequency combination of the desired transfer function vector, then LOCTF contains the desired address.

For first order transfer function vectors, the desired address is equal to the sequential frequency number. The algorithm for calculating the address of higher order vectors is summarized as follows.

STEP 1. Initialization

a) Set sequential frequency number to I=1

b) Initialize SEQ subscript to J=M

c) Clear address by LOCTF=0

d) Initialize number of vectors of order I to C=1

STEP 2· Locate First Vector of Order I, Generate Corresponding Frequency Combination in SEQ

a) Store sequential frequency number SEQ(J)=I

b) Update address by LOCTF=LOCTF+C

c) Calculate number of vectors of order I by C=C*L/I

d) Increment order to I=I+1

e) Decrement SEQ subscript to J=J-1

f) If (J.GT.Ø), frequency combination not complete. Go to Step 2a.

g) Otherwise LOCTF contains address of first transfer function vector of order M. Corresponding frequency combination is complete in SEQ. Go to Step 3.

STEP 3· Compare Desired Frequency Combination to SEQ

a) If each FRQS element equal some SEQ element, algorithm complete. Exit with desired address in LOCTF.

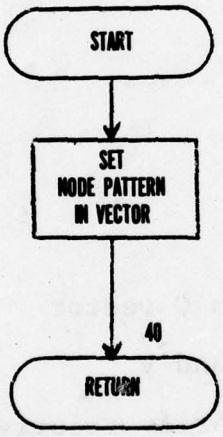
b) Otherwise go to Step 4.

STEP 4: Update Frequency Combination and Address

a) Call subroutine UPFRQ to generate next frequency combination

b) Increment address to LOCTF=LOCTF+1

c) Go to Step 3


START
SET
NODE PATTERN
IN VECTOR
40
RETURN

NAME: QFN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates the function

$$Q_M^K(U,V)$$

for use by subroutine FNCTN

VARIABLES:

K = Length of Q vector

M = Number of nodes V in Q vector

Q = Pattern of nodes U and V

U = Location of nonlinearity relative to base node

V = Location of dependence relative to base node

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

FNCTN

DESCRIPTION:

Subroutine QFN calculates the function Q_M^K (U, V) which determines the pattern of node voltages for each transfer function product computed in subroutine FNCTN. Arguments transmitted to the subroutine are: K, the number of voltages in the transfer function product; M, the number of voltages from node V in the product; U and V, the nodes from which the voltages are derived (relative to the base node of the nonlinear device). The node pattern is returned to the calling program in the integer vector Q.

The function Q_M^K (U, V) insures that each transfer function product calculated by subroutine FNCTN contains K voltages, where the first K-M voltages are derived from node U and the remaining M voltages are derived from node V. The node pattern is stored in the first K elements of the integer vector Q such that the first K-M elements are equal to U and the remaining M elements are equal to V.

$$Q = (\underbrace{U, U, \ldots U}_{K-M \text{ elements}}, \underbrace{V, \ldots\ldots V}_{M \text{ elements}})$$

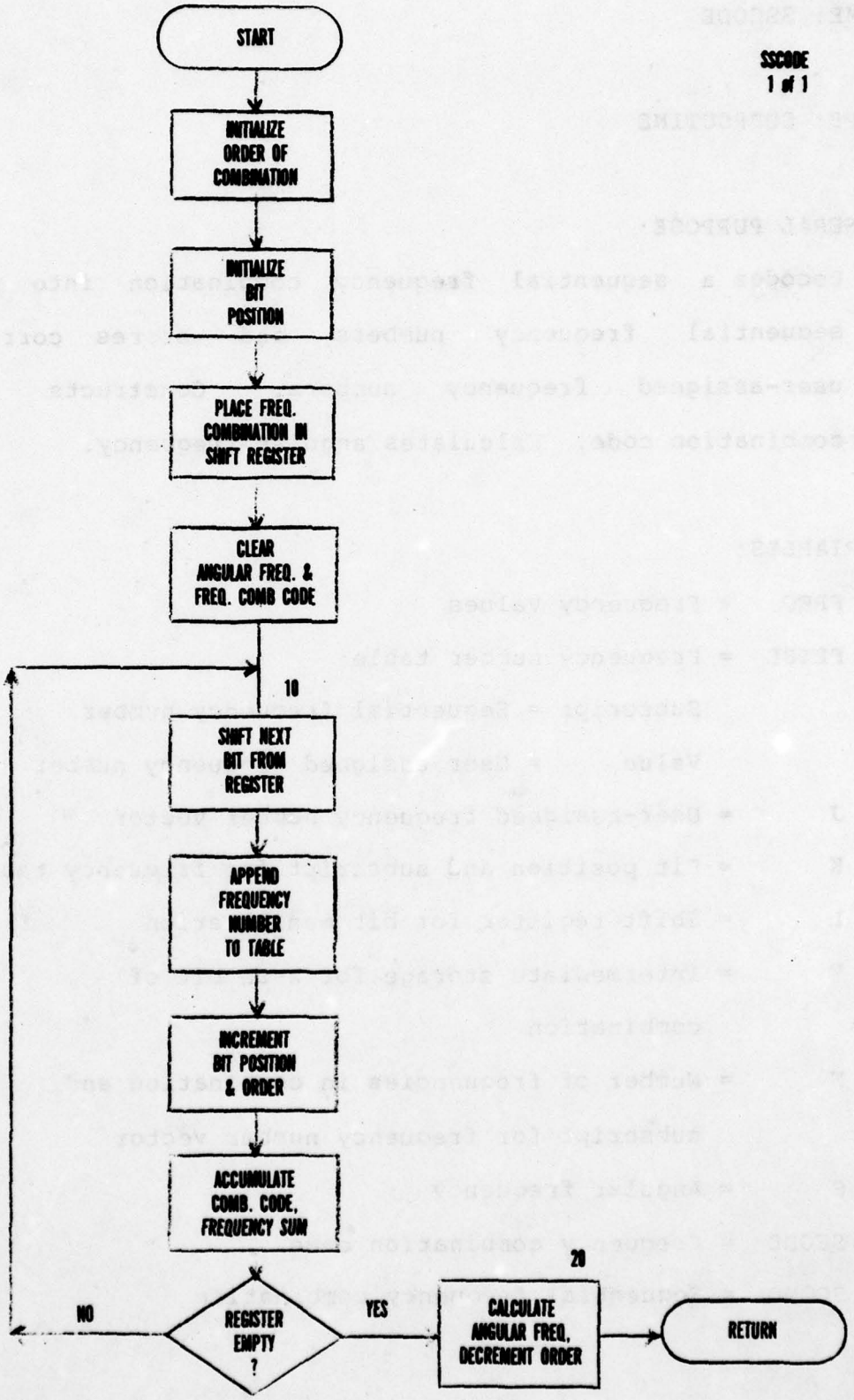
The integers U and V represent the locations of the voltages in each transfer function relative to the base node of the nonlinear device. The base node is represented by 1 in the Q

vector, base+1 is represented by 2, and so on.

As an example, consider a nonlinear current which is a function of the voltages at the base node and base node+1. Then U=1, V=2, and the complete Q_M^K (U, V) function up to sixth order (the largest order allowed in NCAP) is tabulated as follows:

	K=2	K=3	K=4
M=0	$Q_0^2=(1,1)$	$Q_0^3=(1,1,1)$	$Q_0^4=(1,1,1,1)$
M=1	$Q_1^2=(2,1)$	$Q_1^3=(2,1,1)$	$Q_1^4=(2,1,1,1)$
M=2	$Q_2^2=(2,2)$	$Q_2^3=(2,2,1)$	$Q_2^4=(2,2,1,1)$
M=3		$Q_3^3=(2,2,2)$	$Q_3^4=(2,2,2,1)$
M=4			$Q_4^4=(2,2,2,2)$
M=5			
M=6			

	K=5	K=6
M=0	$Q_0^5=(1,1,1,1,1)$	$Q_0^6=(1,1,1,1,1,1)$
M=1	$Q_1^5=(2,1,1,1,1)$	$Q_1^6=(2,1,1,1,1,1)$
M=2	$Q_2^5=(2,2,1,1,1)$	$Q_2^6=(2,2,1,1,1,1)$
M=3	$Q_3^5=(2,2,2,1,1)$	$Q_3^6=(2,2,2,1,1,1)$
M=4	$Q_4^5=(2,2,2,2,1)$	$Q_4^6=(2,2,2,2,1,1)$
M=5	$Q_5^5=(2,2,2,2,2)$	$Q_5^6=(2,2,2,2,2,1)$
M=6		$Q_6^6=(2,2,2,2,2,2)$


START
INITIALIZE ORDER OF COMBINATION
INITIALIZE BIT POSITION
PLACE FREQ. COMBINATION IN SHIFT REGISTER
CLEAR ANGULAR FREQ. & FREQ. COMB CODE
10
SHIFT NEXT BIT FROM REGISTER
APPEND FREQUENCY NUMBER TO TABLE
INCREMENT BIT POSITION & ORDER
ACCUMULATE COMB. CODE, FREQUENCY SUM
REGISTER EMPTY ?
NO
YES
20
CALCULATE ANGULAR FREQ, DECREMENT ORDER
RETURN

NAME: SSCODE

TYPE: SUBROUTINE

GENERAL PURPOSE:

Decodes a sequential frequency combination into component sequential frequency numbers and stores corresponding user-assigned frequency numbers. Constructs frequency combination code. Calculates angular frequency.

VARIABLES:

FREQ = Frequency values

FRTBL = Frequency number table;
Subscript = Sequential frequency number
Value = User assigned frequency number

J = User-assigned frequency number vector

K = Bit position and subscript for frequency table

L = Shift register for bit manipulation

M = Intermediate storage for k-th bit of combination

N = Number of frequencies in combination and subscript for frequency number vector

S = Angular frequency

SCODE = Frequency combination code

SCOMB = Sequential frequency combination

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

CONTRL

DESCRIPTION:

Subroutine SSCODE decodes a sequential frequency combination into its component sequential frequency numbers, stores the corresponding user-assigned frequency numbers, constructs the frequency combination code, and calculates the angular frequency. Arguments transmitted to subroutine SSCODE are: the sequential frequency combination, SCOMB; the frequency number table, FRTBL; and the frequency values FREQ. Arguments returned to the calling program are: the user-assigned frequency numbers in the combination, J; the number of frequencies in the combination, N; the calculated angular frequency, S; and the frequency combination code, SCODE.

The relationship between the sequential frequency combination and the frequency combination code is that if the Kth sequential frequency is in the combination (i.e., the Kth bit of SCOMB is 1), then the user-assigned frequency number FRTBL(K) is to be included in the code (i.e., the FRTBL(K)th bit of SCODE is 1). Similarly the Kth frequency value, FREQ(K), is included in the angular frequency, S, if the Kth sequential frequency appears in the frequency combination.

The method employed by subroutine SSCODE is to extract each

sequential frequency number from SCOMB and, using the frequency number table FRTBL, to combine the corresponding user-assigned frequency numbers and values into the frequency combination code and angular frequency. The sequential frequency numbers are extracted from the combination by using a simulated shift register L, which is initially loaded with the sequential frequency combination SCOMB. A frequency number is extracted from SCOMB by shifting the right-most bit from the register into the integer variable M. At the same time, the integer K maintains a count on the bit position being examined.

If the Kth bit is set (M=1), then the Kth user-assigned frequency number is saved by J(N) = FRTBL(K), the FRTBL(K)th bit of the frequency combination code is set to 1, and FREQ(K) is added to the angular frequency. The calculations are complete when the shift register is empty.

The complete algorithm of subroutine SSCODE is summarized as follows.

STEP 1: Initialization

a) Frequency count: N=1

b) Bit position counter: K=0

c) Load shift register: L=SCOMB

d) Angular frequency: S=0

e) Frequency combination code: SCODE=0

STEP 2: Extract and Store Kth bit of Sequential Combination

a) Increment bit position count K=K+1

b) Shift right-most bit from register and store in M:

M=L

L=L/2

M=M-L*2

c) Save user-assigned frequency number J(N)=FRTBL(K) and increment frequency count by N=N+M

STEP 3: Map Kth Bit to SCODE and S

a) Set FRTBL(K)th bit of frequency combination code according to value of M:

SCODE = SCODE + 2 ** (FRTBL(K) -1) * M

b) Unclude Kth frequency value in angular frequency according to value of M:

S = S + FREQ(K) * FLOAT(M)

STEP 4: Test Shift Register

a) If L=0 (shift register empty), frequency combination code is complete. Go to Step 5.

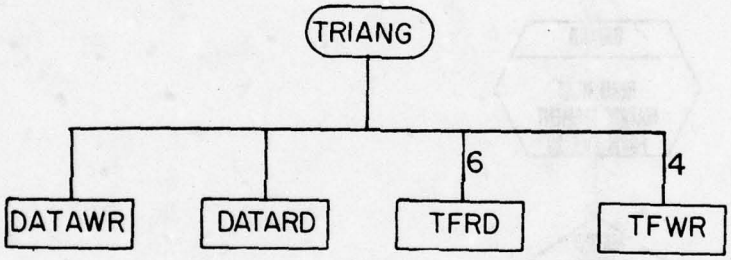
b) Otherwise go to Step 2.

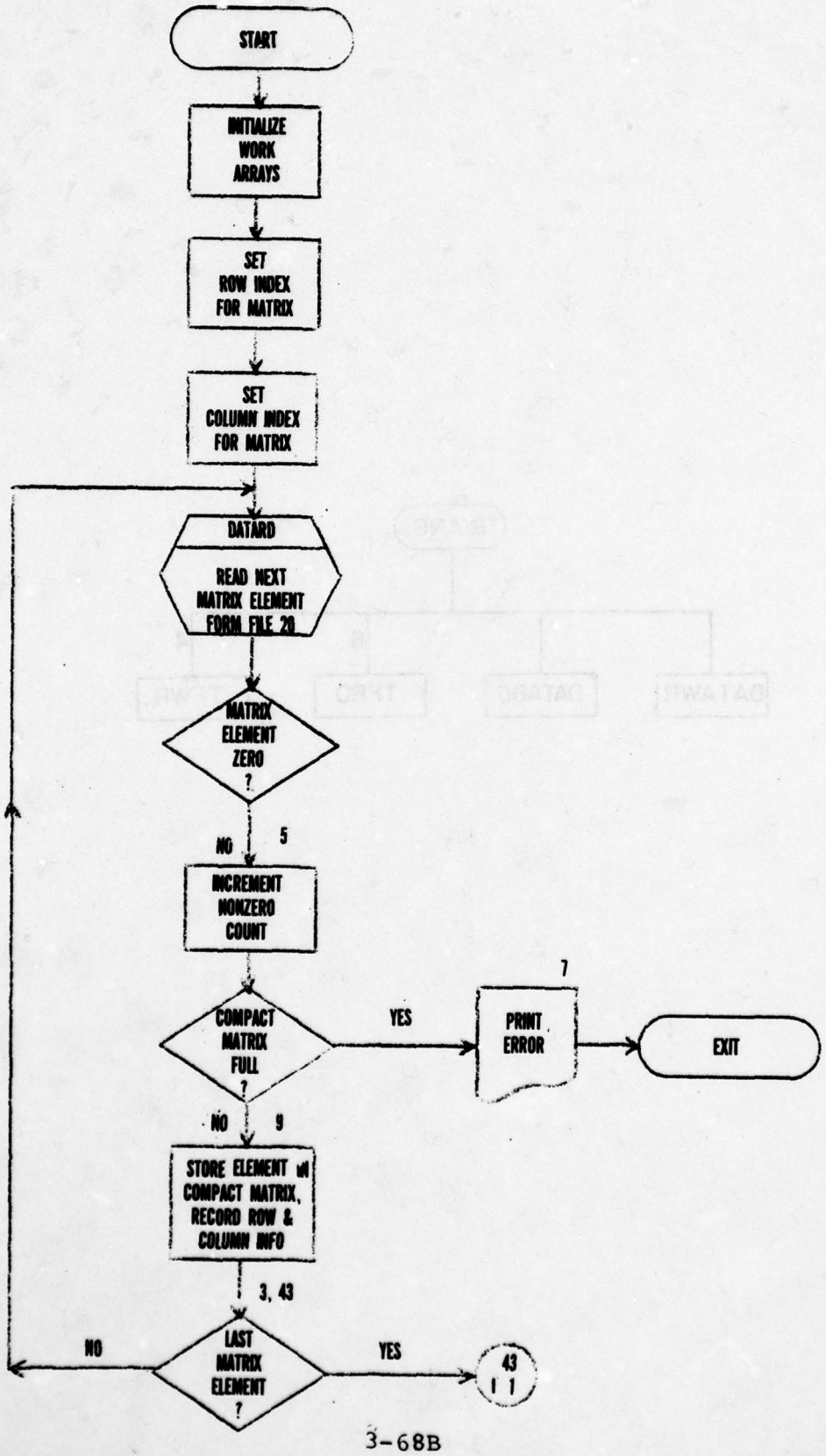
STEP 5: Complete Angular Frequency Calculation

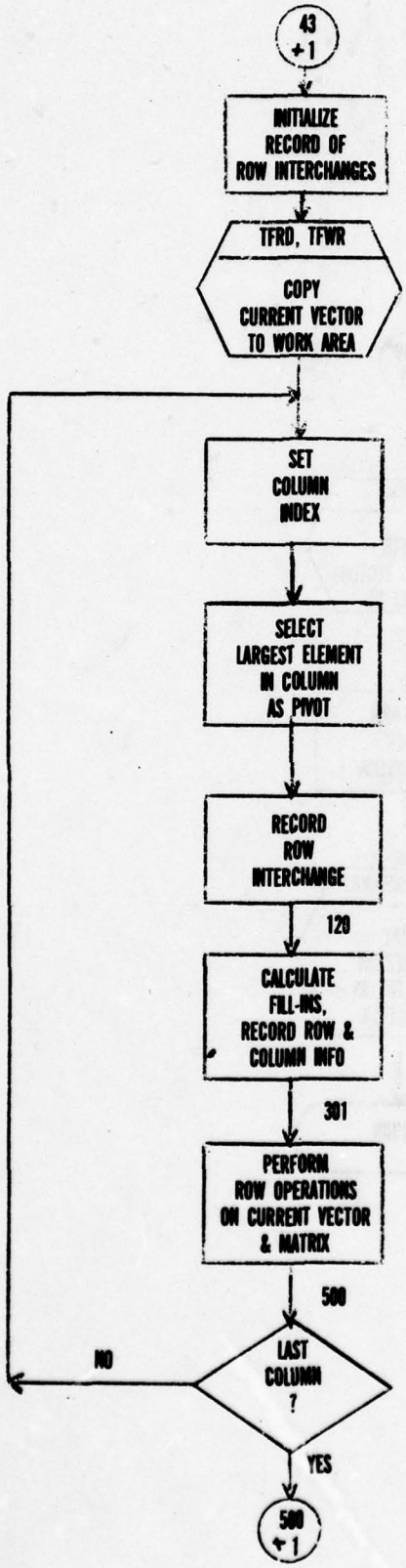
a) Multiply angular frequency by 2

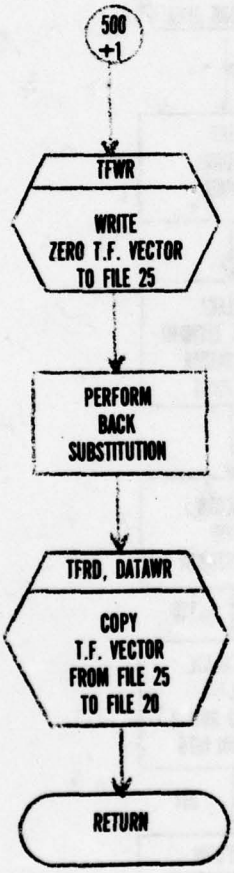
b) Decrement frequency count N=N-1

c) Exit


TRIANG
6
4
DATAWR
DATARD
TFRD
TFWR

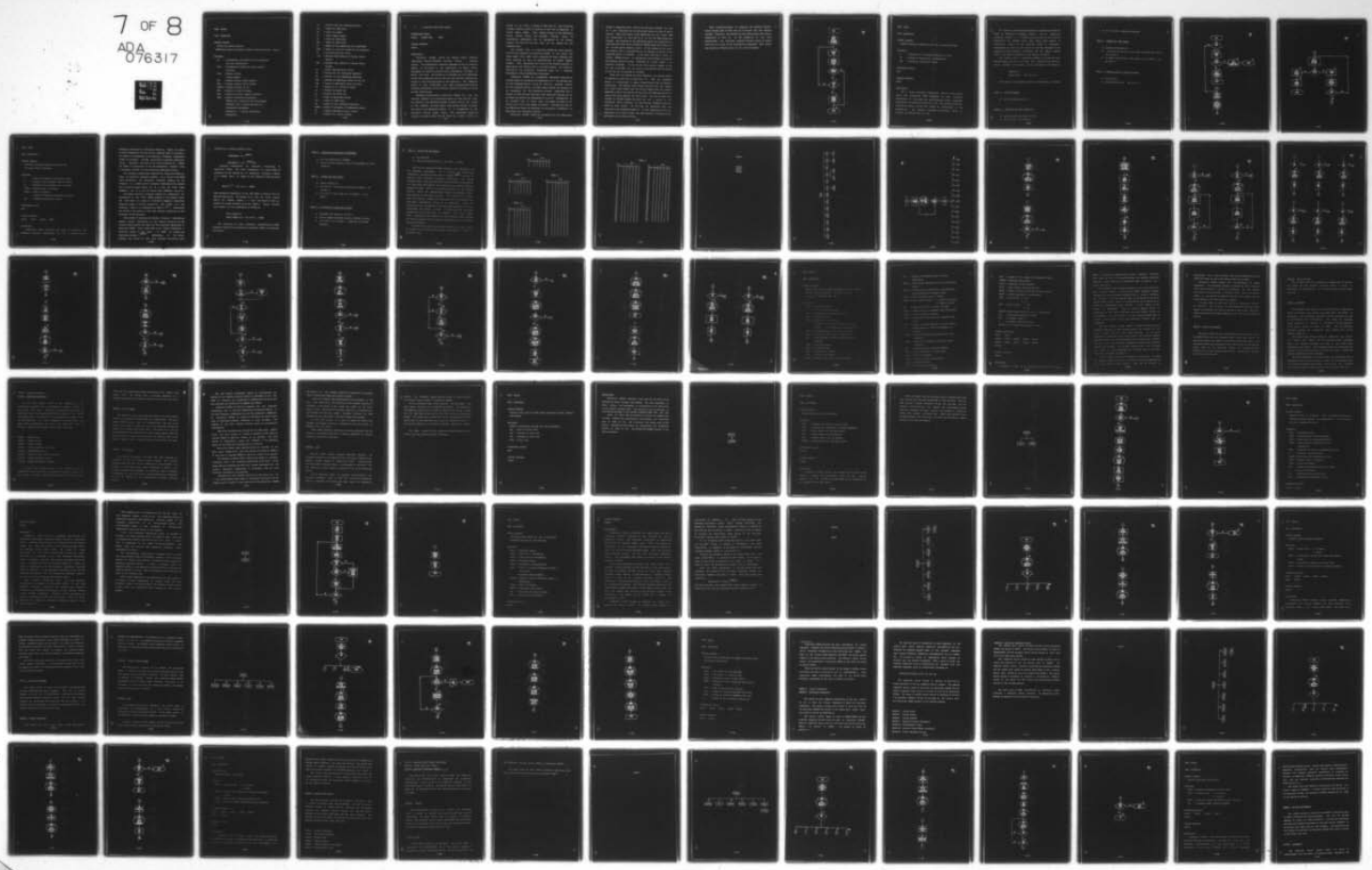

START
INITIALIZE WORK ARRAYS
SET ROW INDEX FOR MATRIX
SET COLUMN INDEX FOR MATRIX
DATARD
READ NEXT MATRIX ELEMENT FORM FILE 20
MATRIX ELEMENT ZERO ?
NO
5
INCREMENT NONZERO COUNT
7
COMPACT MATRIX FULL ?
YES
PRINT ERROR
EXIT
NO
9
STORE ELEMENT in COMPACT MATRIX, RECORD ROW & COLUMN INFO
3, 43
LAST MATRIX ELEMENT ?
NO
YES
43
1 1


43
+1
INITIALIZE RECORD OF ROW INTERCHANGES
TFRD, TFWR
COPY CURRENT VECTOR TO WORK AREA
SET COLUMN INDEX
SELECT LARGEST ELEMENT IN COLUMN AS PIVOT
RECORD ROW INTERCHANGE
120
CALCULATE FILL-INS, RECORD ROW & COLUMN INFO
301
PERFORM ROW OPERATIONS ON CURRENT VECTOR & MATRIX
500
LAST COLUMN ?
NO
YES
500
+1


500
+1
TFWR
WRITE
ZERO T.F. VECTOR
TO FILE 25
PERFORM
BACK
SUBSTITUTION
TFRD, DATAWR
COPY
T.F. VECTOR
FROM FILE 25
TO FILE 20
RETURN

AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

7 OF 8
ADA 076317



NAME: TRIANG

TYPE: SUBROUTINE

GENERAL PURPOSE:

Solves the matrix equation

[Admittance Matrix]*[Transfer Function Vector]=[Current Vector]

VARIABLES:

AM = Intermediate calculation for row operation and back substitution

BIGX = Intermediate storage for pivot element selection

CCUR = Complex current vector element

CUR = Complex current vector element

CUR1 = Complex current vector element

CXZERO = Complex constant (0.,0.)

DIM = Length of U and JU arrays

IBIG = Pointer for pivot element selection

IHOW = Record of pivoting information

IHOW(1)-(N) = Record of row interchanges

IHOW(N+1)-(2N) = Column pointers for elimination information

IHOW(2N+1)... = Column elimination information

II = Column index for triangularization

III = Index for IHOW array

IP = Pivot row number

IPT = Pivot element number

IS = Index for IHOW array

ISS = Index for IHOW array

IT = Number of row operations to be performed

ITOT = Number of fill-ins created by row operation

IU = Row identification vector

JIP = Relative disk address of current vector element

JNR = Relative disk address of current vector element

JU = Column identification vector

K1 = Pointer for row undergoing operation

K2 = Pointer for row undergoing operation

LOC = Address of admittance matrix on file 20

LPE = Length of admittance matrix in words

LT = Length of T.F. vector in words

L1 = Pointer for pivot row

L2 = Pointer for pivot row

N = Order of admittance matrix

NN1 = Index for IHOW array

NR = Number of row undergoing operation

NZERO = Number of nonzeroes in admittance matrix

PE = Complex admittance matrix element

T = Complex T.F. vector element

U = Compacted admittance matrix

SUBROUTINES CALLED:

DATARD DATAWR TFRD TFWR

CALLING PROGRAMS:

PHASE3

DESCRIPTION:

Subroutine TRIANG solves the complex matrix equation [Admittance Matrix]*[Transfer Function Vector] = [Current Vector]. The mathematical technique employed is that of Stoner's Method as described in "The Application of Sparse Matrix Techniques to Logisitcs Related Linear Programming Problems" by Barton and Lloyd. The method is to collapse the full admittance matrix into compacted form so that only the non-zero elements are stored in core and manipulated in the subroutine. The compacted matrix is then transformed into upper triangularized form by Gaussian elimination and the desired transfer functions are found by back substitution.

Arguments transmitted to subroutine TRIANG are: LOC, the starting address of the admittance matrix on disk file 20; LPE, the length of the admittance matrix in words; and LT, the length of the transfer function vector (and current vector) in words. The order of the admittance matrix, N, is transmitted to the subroutine through global common. The admittance matrix is stored by columns on disk file 20, while the current vector is

stored in the first LT words of disk file 25. The calculated transfer function vector is written to disk file 20 beginning at record number DATREC. Each complex element of the admittance matrix, current vector, and transfer function vector is individually addressable and is represented by two decimal numbers, the first for its real part and the second for its imaginary part.

The program code of subroutine TRIANG was taken directly from the main program and subroutine TRIANG of the Barton and Lloyd paper with some minor variations in input technique and pivot strategy as well as modifications to handle complex numbers. This discussion will focus on the changes required to adapt Barton and Lloyd's work to the NCAP program. The reader is directed to the previously referenced paper for a complete description of the mathematical techniques.

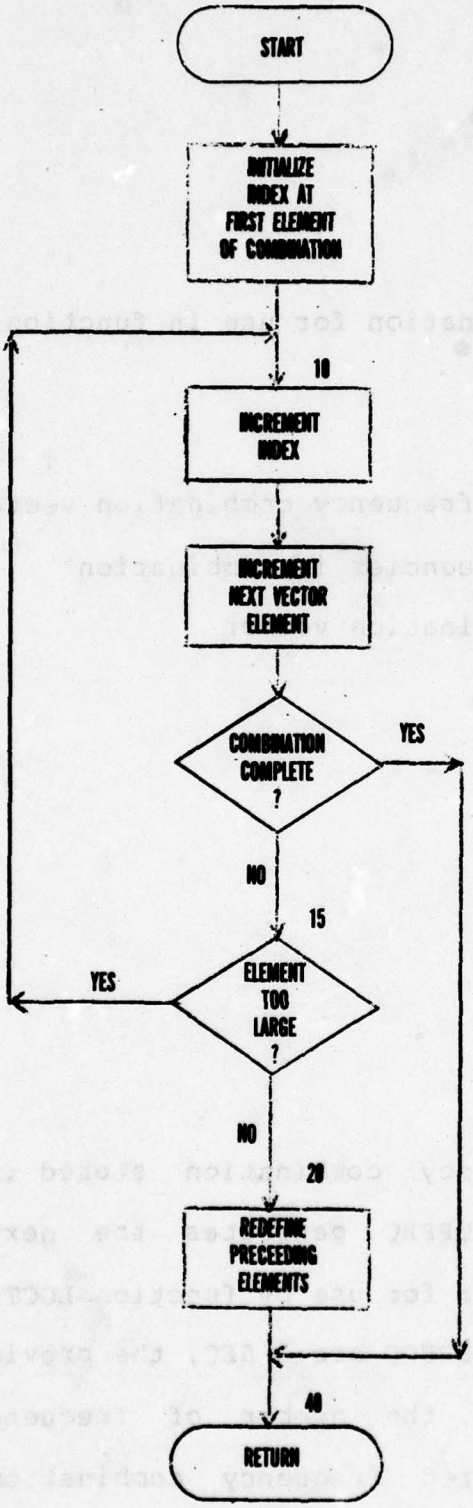
Subroutine TRIANG can accommodate admittance matrices of varying orders by adjusting the dimensions of U, the storage area for the compacted matrix, JU and IU, which are indexing vectors for the compacted matrix, and IHOW, which records the sequence of row operations for the elimination process. Experience with a variety of NCAP circuits has shown that sufficient storage can be allocated by setting the dimensions of U and JU (represented by the variable DIM) at seven times the number of nodes in the circuit and of IU at the number of nodes+1. Although the size of the IHOW array cannot be predicted as accurately, a dimension of IHOW(2*DIM) will generally suffice.

Subroutine TRIANG begins by collapsing the full admittance

matrix to compacted form, storing the non-zero elements by rows in U and recording the row and column indices in the IU and JU vectors. First the nonzero count NZERO and the row count IROW are initialized to zero and the U, IU, JU, and IHOW arrays are cleared. The elements of the admittance matrix are read one at a time from disk file 20 by subroutine DATARD under the control of the row and column indices I and J. If the element is not zero, the nonzero count is incremented and the element is placed in the compacted matrix at U(NZERO). In the event that the U array is overrun (NZERO.GT.DIM), an appropriate error message is printed and program execution stops. Otherwise the column index J is stored at JU(NZERO). If this is the first nonzero element in the Ith row (I.GT.IROW), the row count is incremented and the origin of the Ith row is recorded at IU(IROW).

After the full matiix has been compacted, the current vector is copied to a work area on disk file 25. Then the compacted matrix is transformed to upper triangularized form by Gaussian elimination. The program code follows directly from the Barton and Lloyd paper except that the column element with the largest magnitude (sum of the squares of the real and imaginary parts) is used as the pivot element for that column. The row in which that element is located becomes the pivot row for a series of row operations which eliminate all the non-zero elements below the pivot in that column. As fill-ins are generated they are inserted in the compacted matrix U. At the completion of all row operations for a given column, the same sequence of operations is performed on the current vector.

When triangularization is complete, the transfer function vector storage area on disk file 25 is cleared and the desired transfer functions are derived by back substitution and written temporarily to file 25. At the completion of the back substitution, the resulting transfer function vector is copied from file 25 to file 20 for accessing by subsequent NCAP phases and subroutine TRIANG returns to the calling program.


START
INITIALIZE INDEX AT FIRST ELEMENT OF COMBINATION
10
INCREMENT INDEX
INCREMENT NEXT VECTOR ELEMENT
COMBINATION COMPLETE ?
YES
NO
15
ELEMENT TOO LARGE ?
YES
NO
20
REDEFINE PRECEEDING ELEMENTS
40
RETURN

NAME: UPFRQ

TYPE: SUBROUTINE

GENERAL PURPOSE:

Updates frequency combination for use in function LOCTF

VARIABLES:

N = Subscript for frequency combination vector

NF = Number of frequencies in combination

SEQ = Frequency combination vector

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

LOCTF

DESCRIPTION:

For a given frequency combination stored in the integer vector SEQ, subroutine UPFRQ generates the next frequency combination of like order for use by function LOCTF. Arguments transmitted to subroutine UPFRQ are: SEQ, the previous frecuency combination vector, and NF, the number of frequencies in the combination. The updated frequency combination vector is returned to function LOCTF in SEQ.

The frequency combinations generated by subroutine UPFRQ are based on sequential frequency numbers. The NF sequential frequency numbers in a combination are stored in the first NF elements of the integer vector SEQ, where each element is numerically equal to the frequency number it represents. Frequencies are stored in the vector so that they decrease as the subscript of SEQ increases. Therefore for any vector element SEQ(N), the largest frequency it can contain is NF-N+1.

The method used in subroutine UPFRQ is to find the first element SEQ(N) for N=1,2...NF such that (SEQ(N)+1).LE.(NF-N+1). Having isolated this element, the new combination is formed by:

```
K=SEQ(N)+1
SEQ(I)=K+N-I    for I=1,2...N
```

The algorithm of subroutine UPFRQ is summarized as follows:

STEP 1 Initialization

a) Set SEQ subscript at N=0

STEP 2 Increment and Test Subscript

a) Increment SEQ subscript by N=N+1

b) If (N.LE.NF), go to Step 3.

c) If (N. GT.NF), algorithm complete.

STEP 3: Update and Test SEQ(N)

a) Define SEQ(N)=SEQ(N)+1

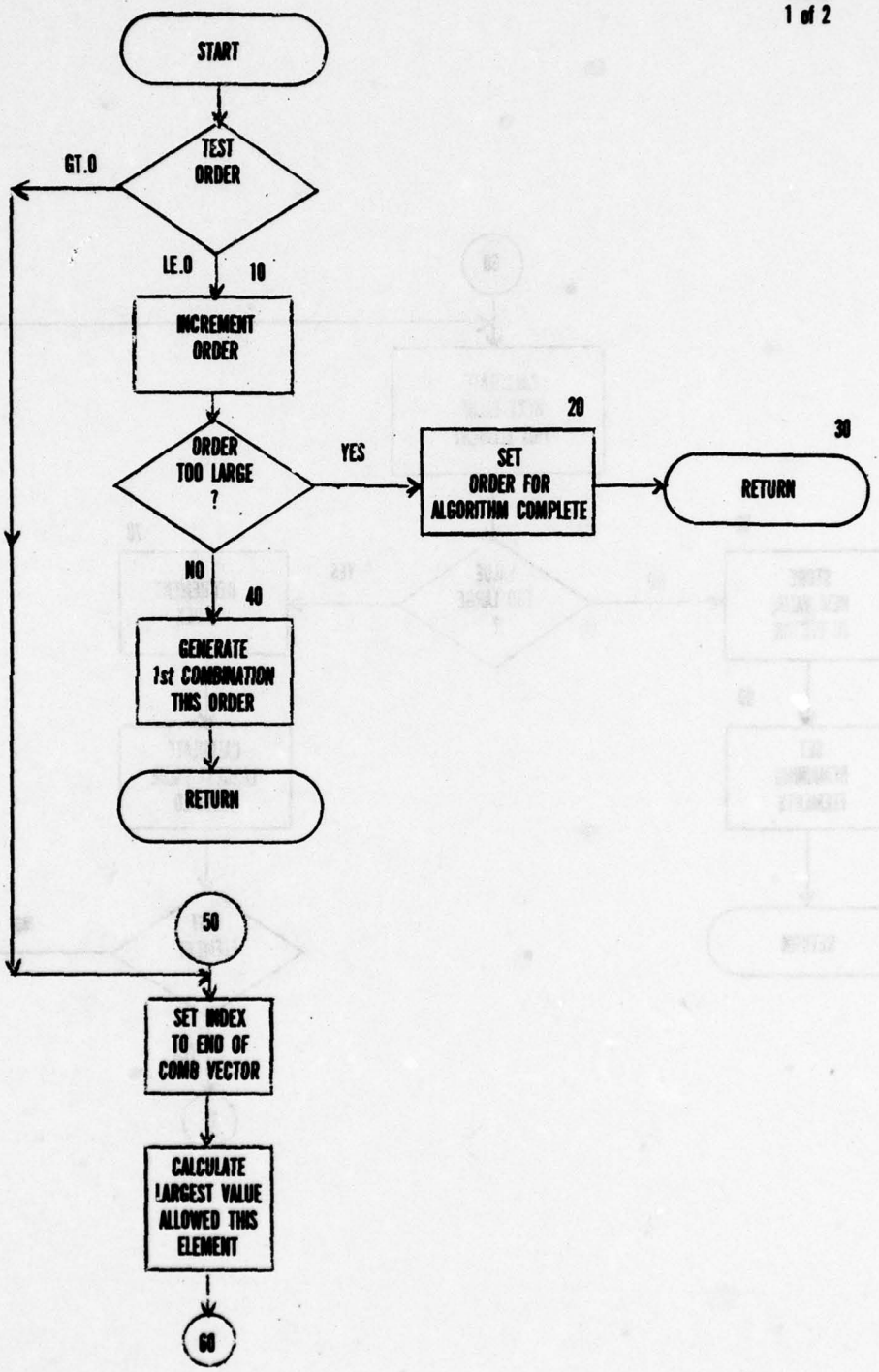
b) If SEQ(N).GT.(NF-N+1), this value too large for SEQ(N). Go to Step 2.

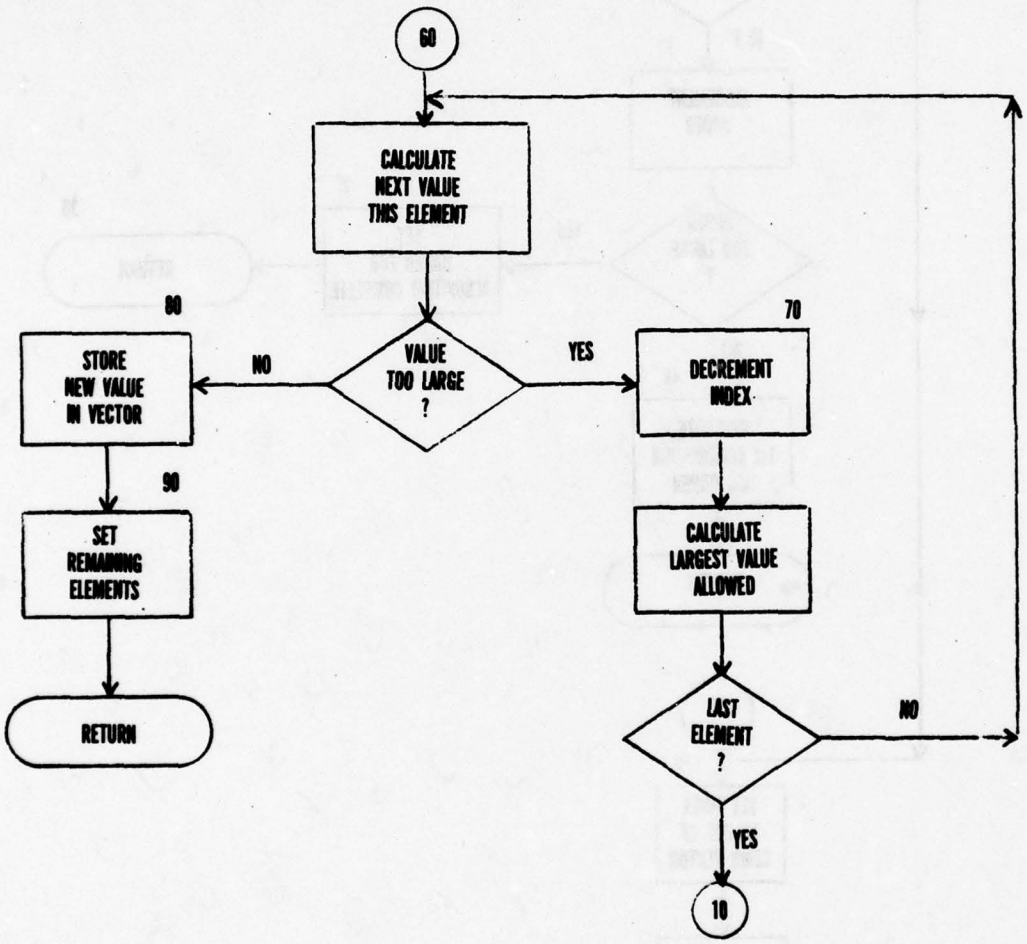
c) If SEQ(N).LE.(NF-N+1), this value O.K. for SEQ(N). Go to Step 4.

STEP 4: Redefine First N Vector Elements

a) Set K=SEQ(N).

b) Define SEQ(I)=K+N-I for I=1,2...N.


START
TEST ORDER
GT.0
LE.0
10
INCREMENT ORDER
ORDER TOO LARGE ?
YES
20
SET ORDER FOR ALGORITHM COMPLETE
30
RETURN
NO
40
GENERATE 1st COMBINATION THIS ORDER
RETURN
50
SET INDEX TO END OF COMB VECTOR
CALCULATE LARGEST VALUE ALLOWED THIS ELEMENT
60


60
CALCULATE NEXT VALUE THIS ELEMENT
VALUE TOO LARGE ?
YES
NO
70
DECREMENT INDEX
80
STORE NEW VALUE IN VECTOR
90
SET REMAINING ELEMENTS
RETURN
CALCULATE LARGEST VALUE ALLOWED
LAST ELEMENT ?
NO
YES
10

NAME: UPSEQ

TYPE: SUBROUTINE

GENERAL PURPOSE:

Generates frequency combination vector and maintains order of analysis

VARIABLES:

I	= Index for frequency combination vector
M	= Largest valid frequency code for SEQ(I)
MM	= Largest valid frequency code in vector
MXORD	= Maximum order of analysis
ORDER	= Order of analysis (length of frequency combination vector)
SEQ	= Frequency combination vector

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

CONTRL CURMC CURNDS PRSET

DESCRIPTION:

Subroutine UPSEQ maintains the order of analysis and generates frequency combinations for use in Phases 3 and 4.

Arguments transmitted to subroutine UPSEQ are: MXORD, the number of input frequencies for the circuit (maximum order of analysis); the number of frequencies in the previous frequency combination (order of analysis); and SEQ, the previous frequency combination vector. Arguments returned to the calling program are: ORDER, the number of frequencies in the new combination (updated order of analysis) and SEQ, the new frequency combination vector.

The frequency combinations generated by subroutine UPSEQ are based on sequential frequency numbers. For a circuit with MXORD input frequencies, the sequential frequency numbers are the integers 1,2,...MXORD and the frequency combinations are ordered sets of these integers taken one at a time for first order (ORDER=1), two at a time for second order (ORDER=2), and so on.

The ORDER sequential frequency numbers in a combination are represented by the first ORDER elements of the integer vector SEQ. Each SEQ(I) is a power of 2 derived by mapping a sequential frequency number to the bit position of like number (i.e. the frequency number N is represented by $SEQ(I) = 2^{N-1}$). Frequencies are stored in the vector so that they increase numerically as the subscript of SEQ increases.

This method of encoding and storing frequency combinations imposes certain limitations on the numeric values of the SEQ elements which provide the basis for the algorithm implemented in subroutine UPSEQ. First, since each vector element represents a frequency number in the range 1 to MXORD, no element may numerically exceed $2^{MXORD-1}$. Furthermore, since the vector elements are stored so that they increase numerically, each

element has a largest possible value:

$$SEQ(ORDER) \text{ .LE. } 2^{MXORD-1}$$

$$SEQ(ORDER-1) \text{ .LE. } 2^{MXORD-2}, \text{etc.}$$

Frequency combinations are generated recursively by subroutine UPSEQ. The first combination of a given ORDER is defined to be the ordered set of sequential frequency numbers (1,2...ORDER) which is mapped to the elements of SEQ according to:

$$SEQ(I)=2^{I-1} \quad \text{for } I=1,2,...ORDER$$

Each successive combination of the same ORDER is derived from the previous SEQ vector. The method is to find the first element SEQ(I) for I=ORDER, ORDER-1,...1 such that SEQ(I)*2 does not exceed the largest possible value for SEQ(I). Having isolated this element, the new combination is formed by:

$$SEQ(I)=SEQ(I)*2$$
$$SEQ(J)=SEQ(J-1)*2 \quad \text{for } J=I+1,...ORDER$$

The algorithm by which successive combinations of ORDER frequency numbers are generated by subroutine UPSEQ is summarized as follows:

STEP 1: Initialize Algorithm at SEQ(ORDER)

a) Set SEQ subscript at I=ORDER.

b) Define largest possible value for SEQ(ORDER) as M=2** (MXORD-1).

STEP 2: Update and Test SEQ(I)

a) Define J=SEQ(I)*2.

b) If (J.GT.M), this value too large for SEQ(I). Go to Step 3.

c) If (J.LE.M), this value O.K. for SEQ(I). Go to Step 4.

STEP 3: Move Search to Next SEQ Element

a) Decrement SEQ subscript by I=I-1.

b) Define largest possible value for SEQ(I) as M=M/2.

c) If (I.GT.0), go to Step 2. Otherwise this ORDER complete.

STEP 4· Define New SEQ Vector

a) Set SEQ(I)=J.

b) Define SEQ(J)=SEQ(J-1)*2 for J=I+1,...ORDER.

Each call to subroutine UPSEQ results in the generation of one frequency combination. For a circuit with MXORD input frequencies, subroutine UPSEQ must be invoked $2^{MXORD}-1$ times to generate every possible frequency combination. The integer variable ORDER, which primarily defines the number of frequencies in a particular combination, also controls the iterative calls to subroutine UPSEQ. ORDER is initialized to zero by the calling program, tested and appropriately updated by subroutine UPSEQ, then retested by the calling program. For ORDER=0,1,2...MXORD, both the calling program and subroutine UPSEQ are alerted that additional combinations remain to be generated. After all combinations for a particular ORDER have been generated, ORDER is incremented and tested in UPSEQ. If ORDER.LE.MXORD, the first combination for the new ORDER is created. When ORDER exceed MXORD, subroutine UPSEQ returns the value ORDER=-1 which signals the calling program that all possible combinations for all orders have been generated.

A tabulation of all frequency combinations for every order of analysis up to MXORD=6 (the largest order allowed in NCAP), is presented below:

MXORD = 2

NO.	SEQ 1	2	3	4	5	6
1	1	0	0	0	0	0
2	2	0	0	0	0	0
3	2	1	0	0	0	0

MXORD = 3

NO.	SEQ 1	2	3	4	5	6
1	1	0	0	0	0	0
2	2	0	0	0	0	0
3	3	0	0	0	0	0
4	2	1	0	0	0	0
5	3	1	0	0	0	0
6	3	2	0	0	0	0
7	3	2	1	0	0	0

MXORD = 4

NO.	SEQ 1	2	3	4	5	6
1	1	0	0	0	0	0
2	2	0	0	0	0	0
3	3	0	0	0	0	0
4	4	0	0	0	0	0
5	2	1	0	0	0	0
6	3	1	0	0	0	0
7	4	1	0	0	0	0
8	3	2	0	0	0	0
9	4	2	0	0	0	0
10	4	3	0	0	0	0
11	3	2	1	0	0	0
12	4	2	1	0	0	0
13	4	3	1	0	0	0
14	4	3	2	0	0	0
15	4	3	2	1	0	0

MXORD = 5

NO.	SEQ 1	2	3	4	5	6
1	1	0	0	0	0	0
2	2	0	0	0	0	0
3	3	0	0	0	0	0
4	4	0	0	0	0	0
5	5	0	0	0	0	0
6	2	1	0	0	0	0
7	3	1	0	0	0	0
8	4	1	0	0	0	0
9	5	1	0	0	0	0
10	3	2	0	0	0	0
11	4	2	0	0	0	0
12	5	2	0	0	0	0
13	4	3	0	0	0	0
14	5	3	0	0	0	0
15	5	4	0	0	0	0
16	3	2	1	0	0	0
17	4	2	1	0	0	0
18	5	2	1	0	0	0
19	4	3	1	0	0	0
20	5	3	1	0	0	0
21	5	4	1	0	0	0
22	4	3	2	0	0	0
23	5	3	2	0	0	0
24	5	4	2	0	0	0
25	5	4	3	0	0	0
26	4	3	2	1	0	0
27	5	3	2	1	0	0
28	5	4	2	1	0	0
29	5	4	3	1	0	0
30	5	4	3	2	0	0
31	5	4	3	2	1	0

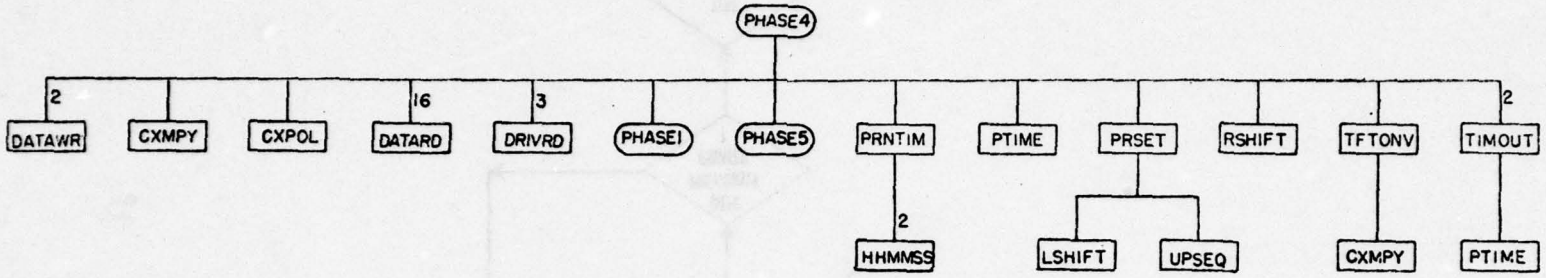
MXORD = 6

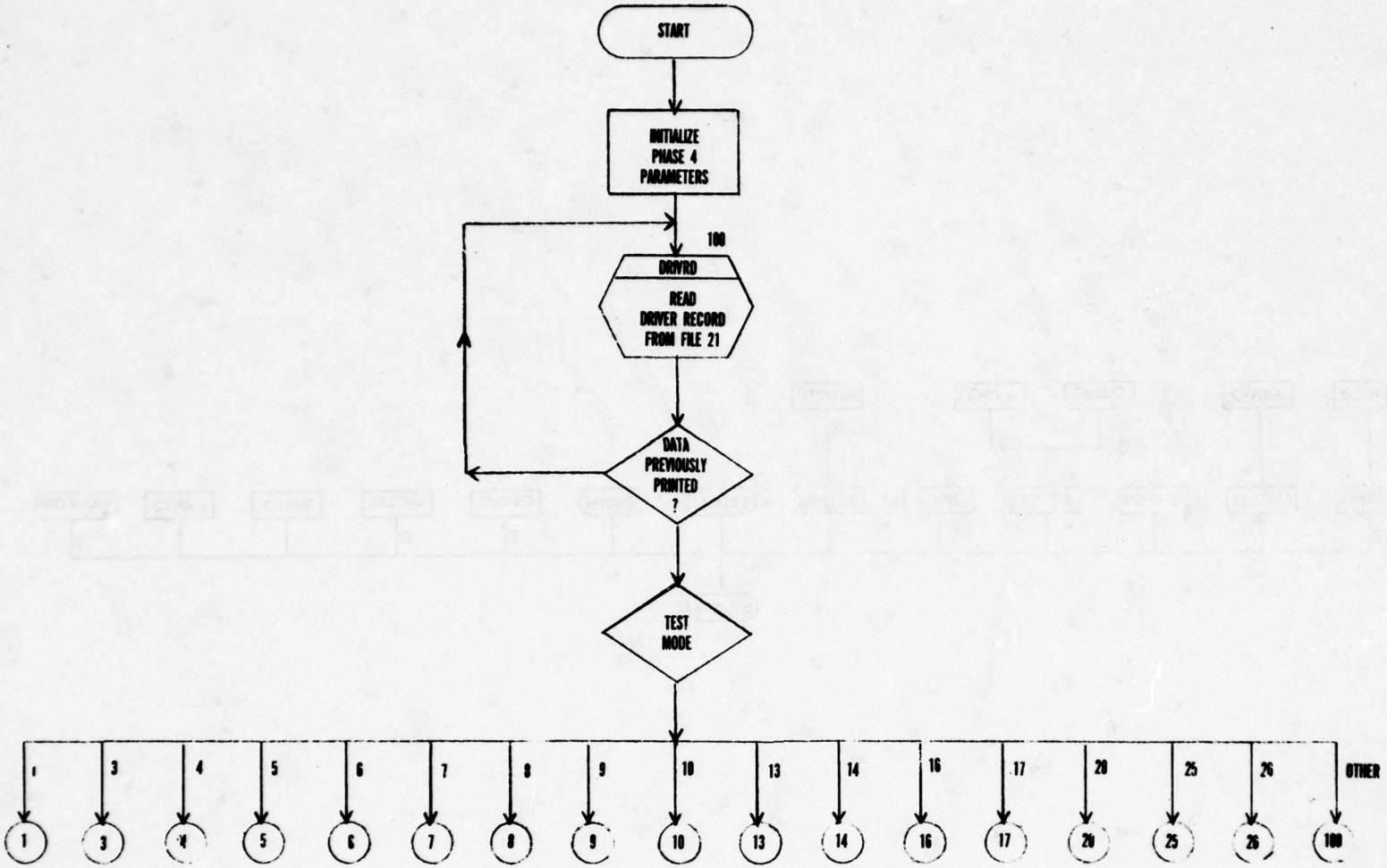
NO.	SEQ 1	2	3	4	5	6
1	1	0	0	0	0	0
2	2	0	0	0	0	0
3	3	0	0	0	0	0
4	4	0	0	0	0	0
5	5	0	0	0	0	0
6	6	0	0	0	0	0
7	2	1	0	0	0	0
8	3	1	0	0	0	0
9	4	1	0	0	0	0
10	5	1	0	0	0	0
11	6	1	0	0	0	0
12	3	2	0	0	0	0
13	4	2	0	0	0	0
14	5	2	0	0	0	0
15	6	2	0	0	0	0
16	4	3	0	0	0	0
17	5	3	0	0	0	0
18	6	3	0	0	0	0
19	5	4	0	0	0	0
20	6	4	0	0	0	0
21	6	5	0	0	0	0
22	3	2	1	0	0	0
23	4	2	1	0	0	0
24	5	2	1	0	0	0
25	6	2	1	0	0	0
26	4	3	1	0	0	0
27	5	3	1	0	0	0
28	6	3	1	0	0	0
29	5	4	1	0	0	0
30	6	4	1	0	0	0
31	6	5	1	0	0	0
32	4	3	2	0	0	0

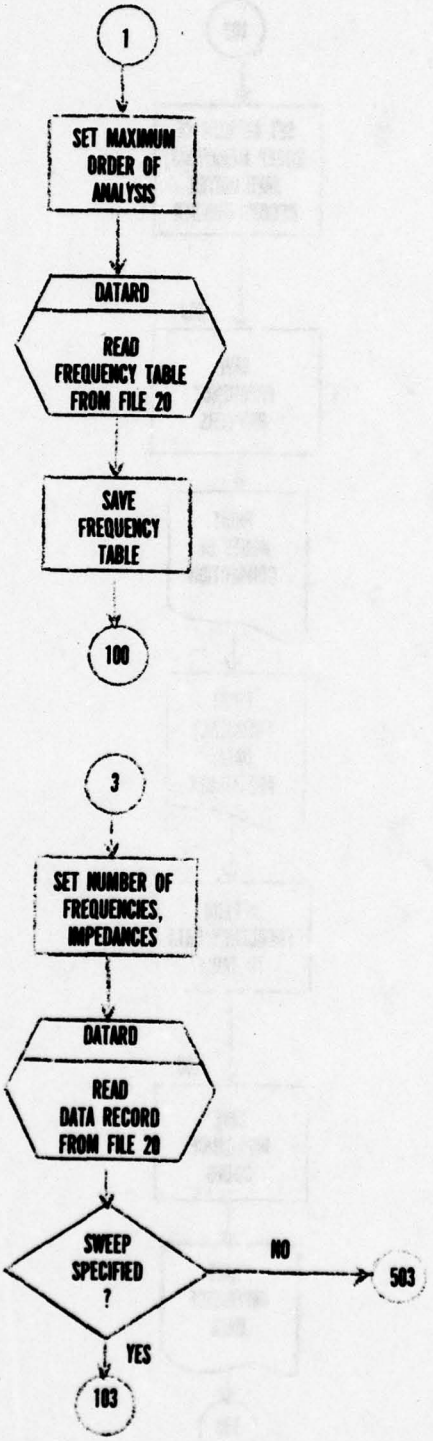
NO.	SEQ 1	2	3	4	5	6
33	5	3	2	0	0	0
34	6	3	2	0	0	0
35	5	4	2	0	0	0
36	6	4	2	0	0	0
37	6	5	2	0	0	0
38	5	4	3	0	0	0
39	6	4	3	0	0	0
40	6	5	3	0	0	0
41	6	5	4	0	0	0
42	4	3	2	1	0	0
43	5	3	2	1	0	0
44	6	3	2	1	0	0
45	5	4	2	1	0	0
46	6	4	2	1	0	0
47	6	5	2	1	0	0
48	5	4	3	1	0	0
49	6	4	3	1	0	0
50	6	5	3	1	0	0
51	6	5	4	1	0	0
52	5	4	3	2	0	0
53	6	4	3	2	0	0
54	6	5	3	2	0	0
55	6	5	4	2	0	0
56	6	5	4	3	0	0
57	5	4	3	2	1	0
58	6	4	3	2	1	0
59	6	5	3	2	1	0
60	6	5	4	2	1	0
61	6	5	4	3	1	0
62	6	5	4	3	2	0
63	6	5	4	3	2	1

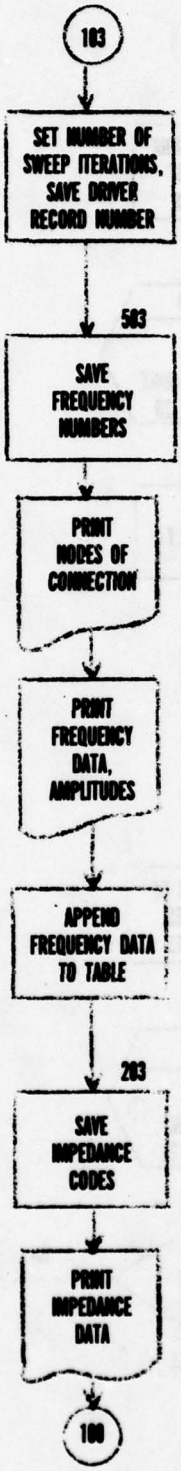
PHASE4

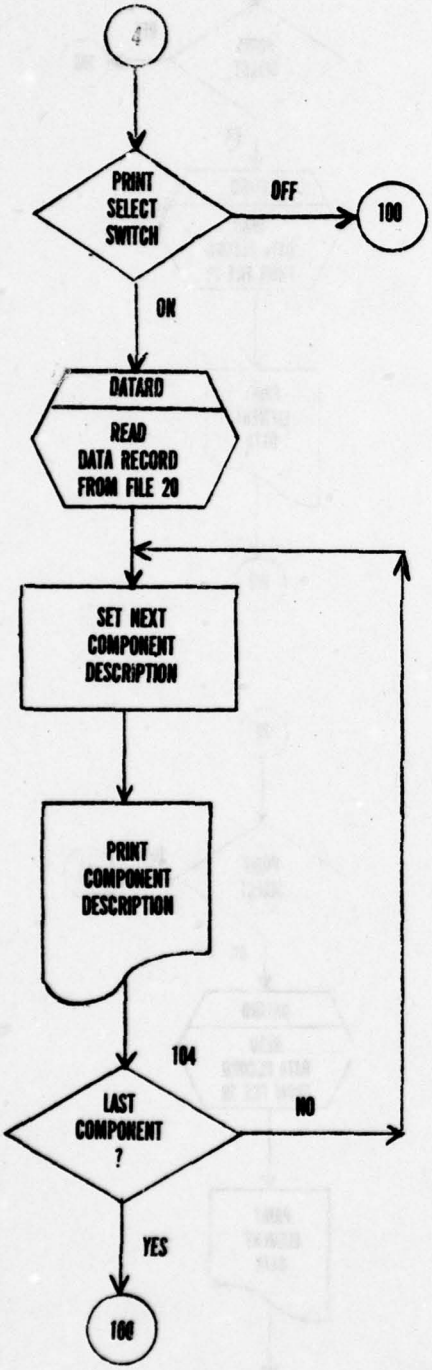
HHMMSS
PRNTIM
PRSET
TFTONV

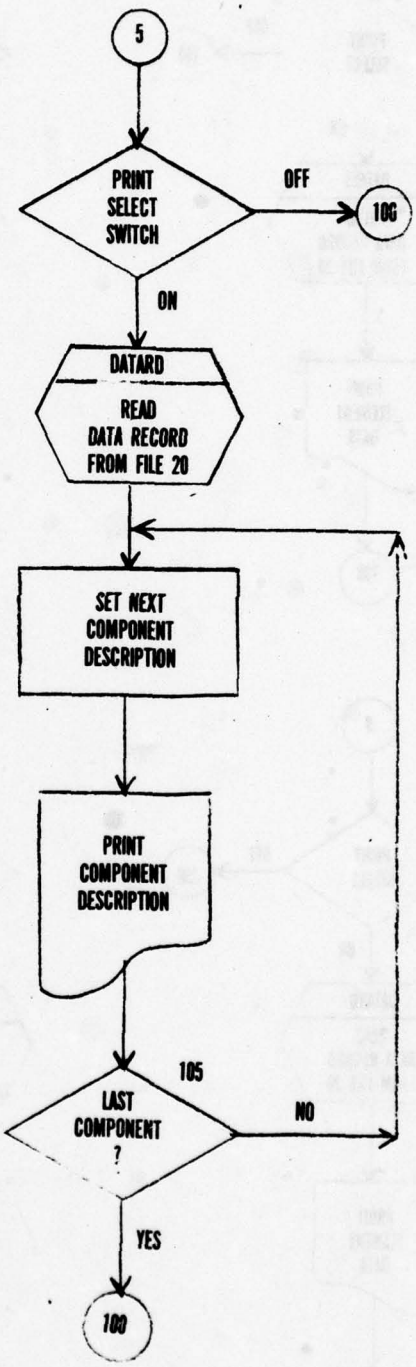

PHASE4
2
DATAWR
CXMPY
CXPOL
16
DATARD
3
DRIVRD
PHASE1
PHASE5
PRNTIM
2
HHMMSS
PTIME
PRSET
LSHIFT
UPSEQ
RSHIFT
TFTONV
CXMPY
2
TIMOUT
PTIME

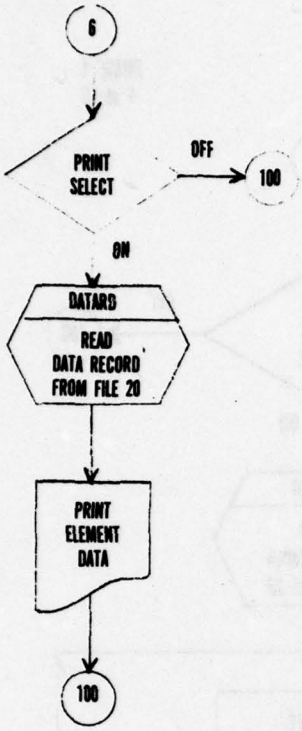

START
INITIALIZE PHASE 4 PARAMETERS
100
DRIVRD
READ DRIVER RECORD FROM FILE 21
DATA PREVIOUSLY PRINTED ?
TEST MODE
1
3
4
5
6
7
8
9
10
13
14
16
17
20
25
26
OTHER
1
3
4
5
6
7
8
9
10
13
14
16
17
20
25
26
100

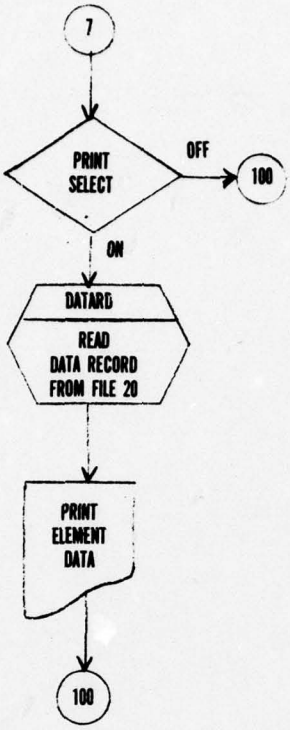

1
SET MAXIMUM ORDER OF ANALYSIS
DATARD
READ FREQUENCY TABLE FROM FILE 20
SAVE FREQUENCY TABLE
100
3
SET NUMBER OF FREQUENCIES, IMPEDANCES
DATARD
READ DATA RECORD FROM FILE 20
SWEEP SPECIFIED ?
NO
503
YES
103

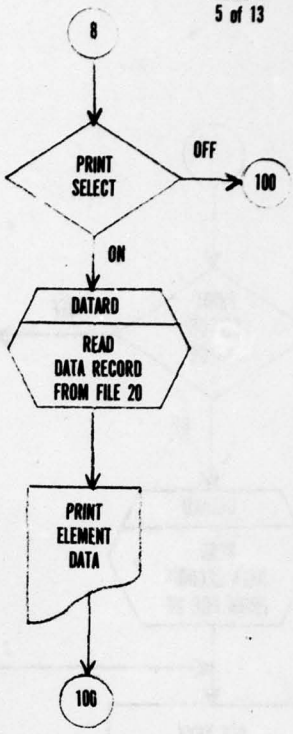

103
SET NUMBER OF SWEEP ITERATIONS, SAVE DRIVER RECORD NUMBER
503
SAVE FREQUENCY NUMBERS
PRINT NODES OF CONNECTION
PRINT FREQUENCY DATA, AMPLITUDES
APPEND FREQUENCY DATA TO TABLE
203
SAVE IMPEDANCE CODES
PRINT IMPEDANCE DATA
100

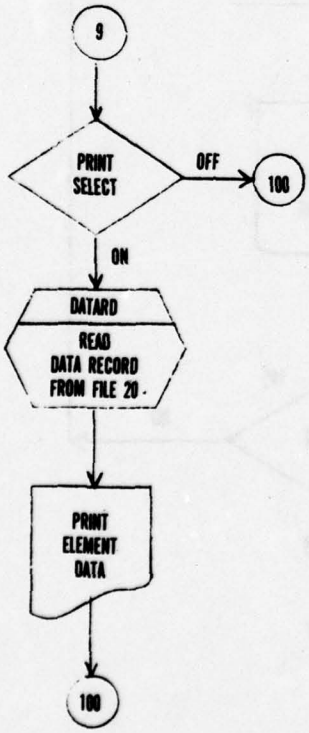

4
PRINT SELECT SWITCH
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
SET NEXT COMPONENT DESCRIPTION
PRINT COMPONENT DESCRIPTION
104
LAST COMPONENT ?
NO
YES
100

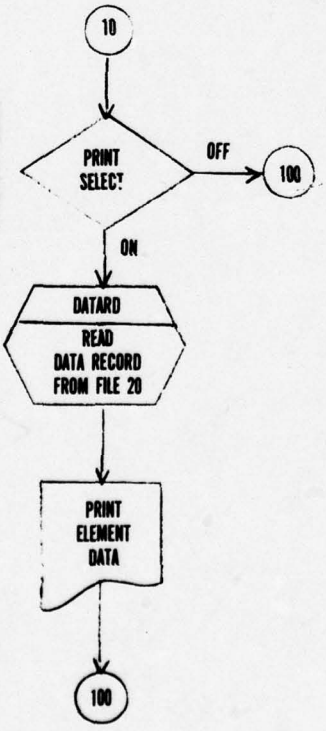

5
PRINT SELECT SWITCH
OFF
10G
ON
DATARD
READ DATA RECORD FROM FILE 20
SET NEXT COMPONENT DESCRIPTION
PRINT COMPONENT DESCRIPTION
105
LAST COMPONENT ?
NO
YES
100

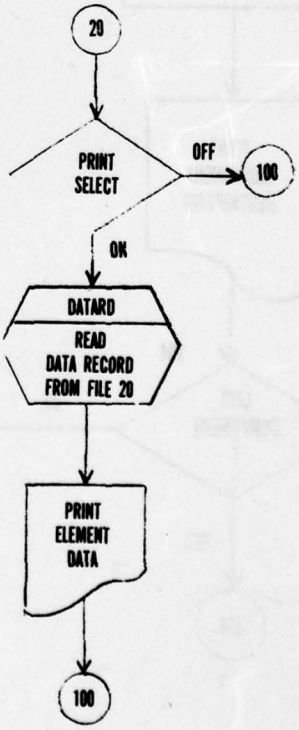

6
PRINT SELECT
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
PRINT ELEMENT DATA
100

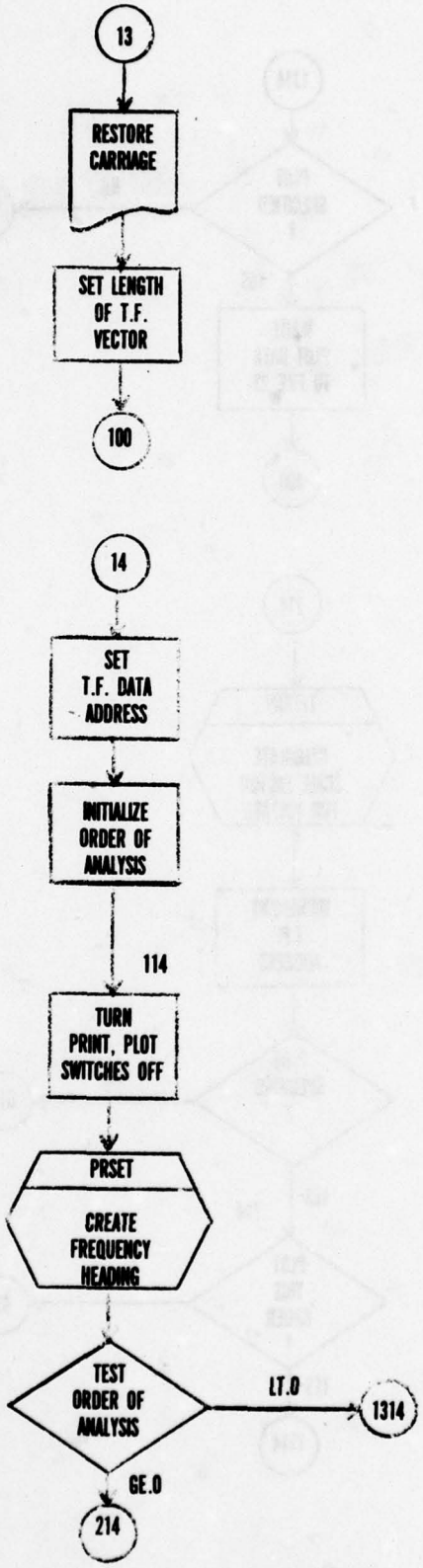

7
PRINT SELECT
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
PRINT ELEMENT DATA
100

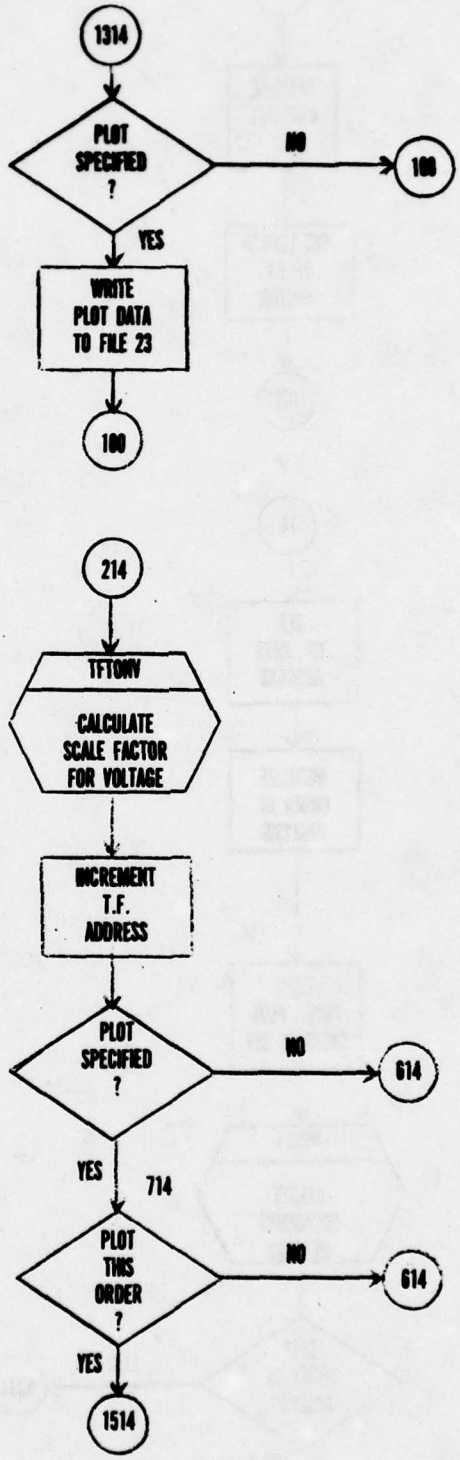

8
PRINT SELECT
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
PRINT ELEMENT DATA
100

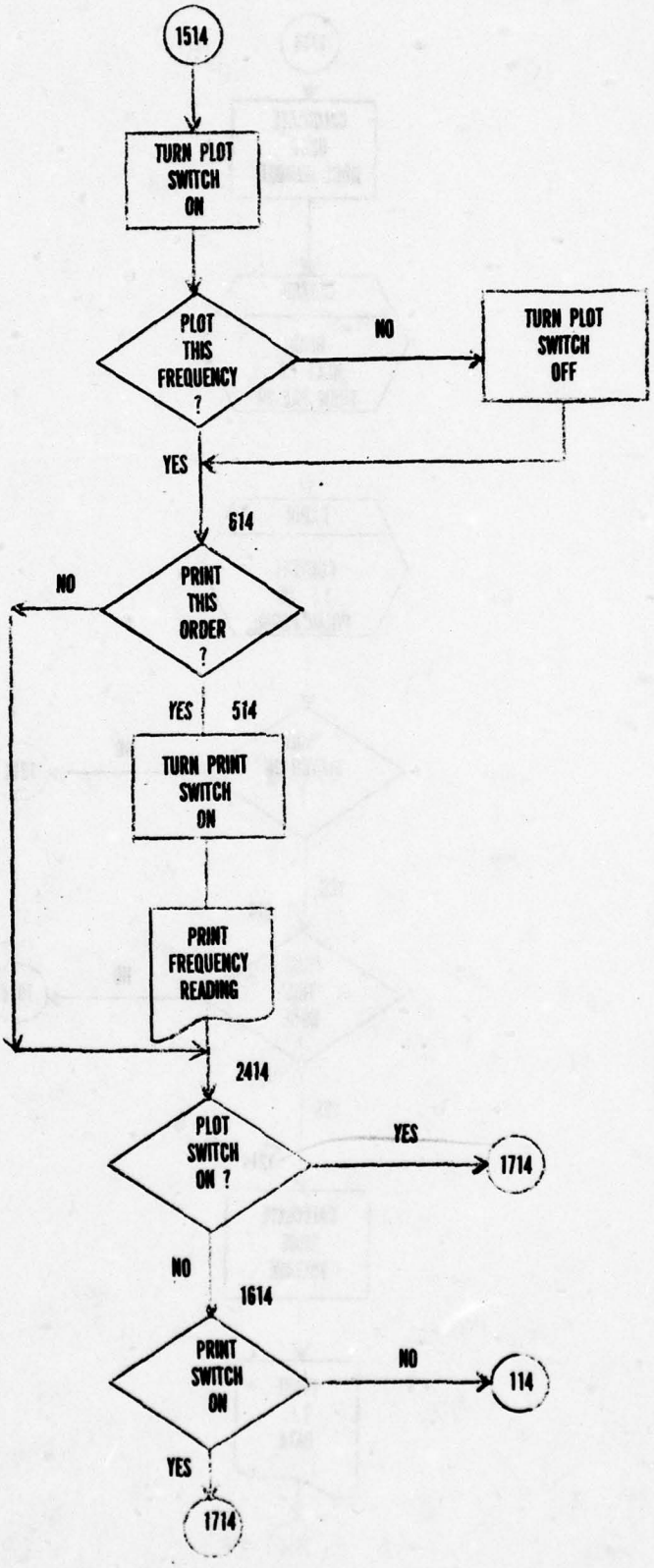

9
PRINT SELECT
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
PRINT ELEMENT DATA
100

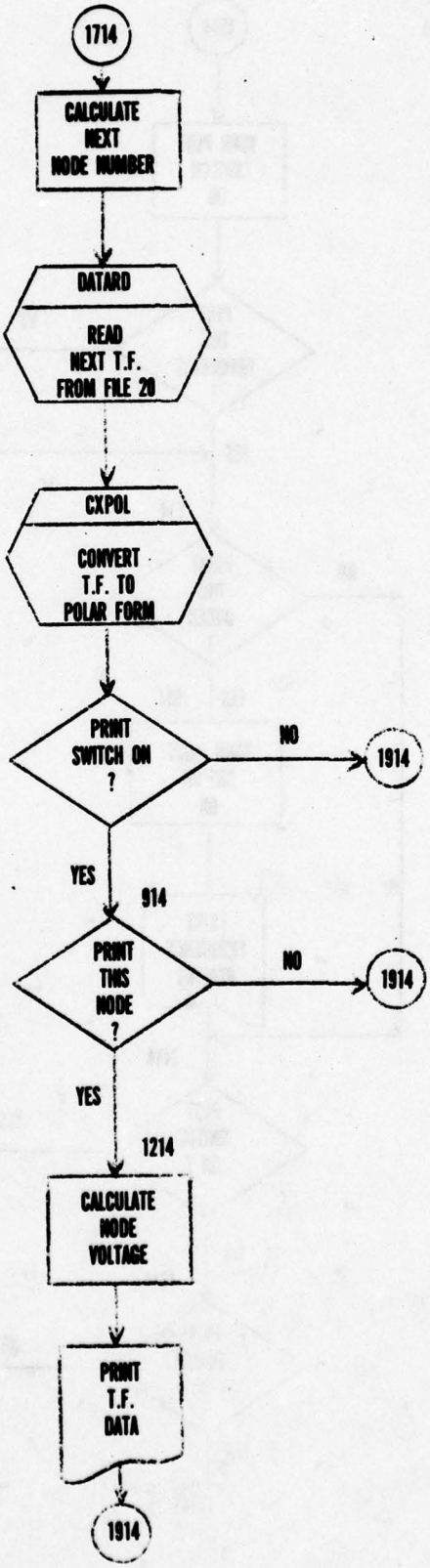

10
PRINT SELECT
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
PRINT ELEMENT DATA
100

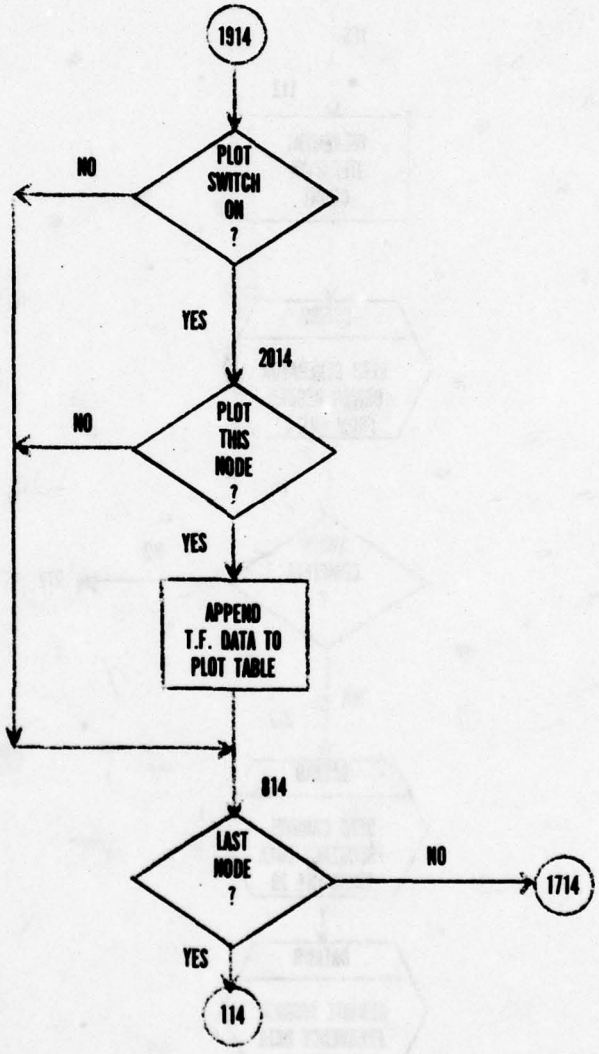

20
PRINT SELECT
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
PRINT ELEMENT DATA
100

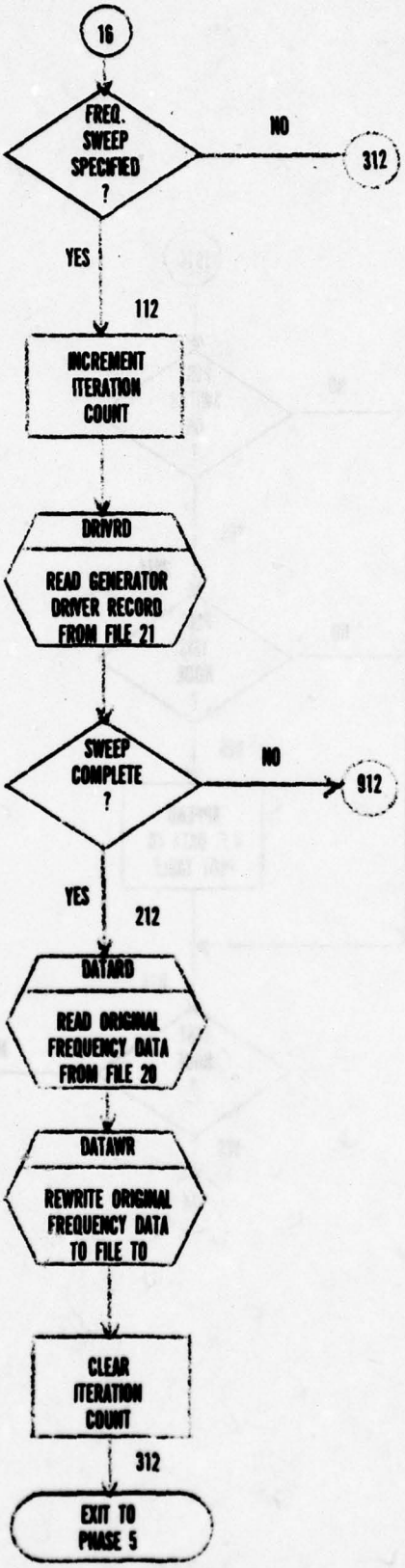

13
RESTORE CARRIAGE
SET LENGTH OF T.F. VECTOR
100
14
SET T.F. DATA ADDRESS
INITIALIZE ORDER OF ANALYSIS
114
TURN PRINT, PLOT SWITCHES OFF
PRSET
CREATE FREQUENCY HEADING
TEST ORDER OF ANALYSIS
LT.0
1314
GE.0
214

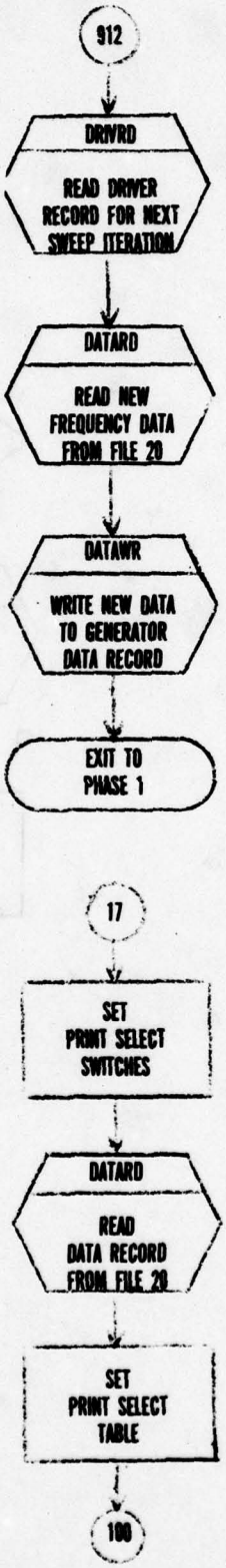

1314
PLOT SPECIFIED ?
NO
100
YES
WRITE PLOT DATA TO FILE 23
100
214
TFTONV
CALCULATE SCALE FACTOR FOR VOLTAGE
INCREMENT T.F. ADDRESS
PLOT SPECIFIED ?
NO
614
YES
714
PLOT THIS ORDER ?
NO
614
YES
1514

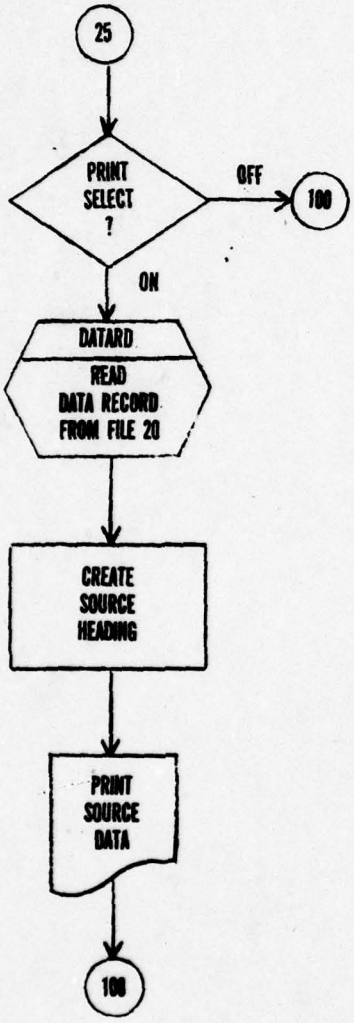

1514
TURN PLOT SWITCH ON
PLOT THIS FREQUENCY ?
NO
TURN PLOT SWITCH OFF
YES
614
PRINT THIS ORDER ?
NO
YES
514
TURN PRINT SWITCH ON
PRINT FREQUENCY READING
2414
PLOT SWITCH ON ?
YES
1714
NO
1614
PRINT SWITCH ON
NO
114
YES
1714

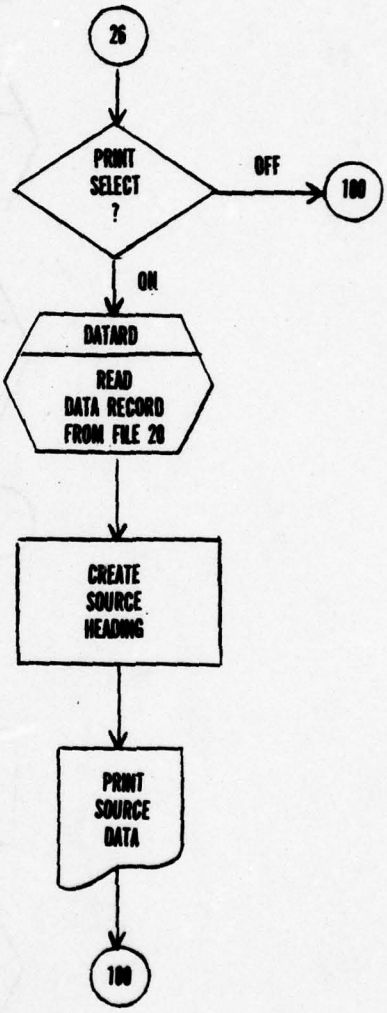

1714
CALCULATE NEXT NODE NUMBER
DATARD
READ NEXT T.F. FROM FILE 20
CXPOL
CONVERT T.F. TO POLAR FORM
PRINT SWITCH ON ?
NO
1914
YES
914
PRINT THIS NODE ?
NO
1914
YES
1214
CALCULATE NODE VOLTAGE
PRINT T.F. DATA
1914


1914
PLOT SWITCH ON ?
NO
YES
2014
PLOT THIS NODE ?
NO
YES
APPEND T.F. DATA TO PLOT TABLE
814
LAST NODE ?
NO
1714
YES
114


16
FREQ. SWEEP SPECIFIED ?
NO
312
YES
112
INCREMENT ITERATION COUNT
DRIVRD
READ GENERATOR DRIVER RECORD FROM FILE 21
SWEEP COMPLETE ?
NO
912
YES
212
DATARD
READ ORIGINAL FREQUENCY DATA FROM FILE 20
DATAWR
REWRITE ORIGINAL FREQUENCY DATA TO FILE TO
CLEAR ITERATION COUNT
312
EXIT TO PHASE 5


912
DRIVRD
READ DRIVER RECORD FOR NEXT SWEEP ITERATION
DATARD
READ NEW FREQUENCY DATA FROM FILE 20
DATAWR
WRITE NEW DATA TO GENERATOR DATA RECORD
EXIT TO PHASE 1
17
SET PRINT SELECT SWITCHES
DATARD
READ DATA RECORD FROM FILE 20
SET PRINT SELECT TABLE
100


25
PRINT SELECT ?
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
CREATE SOURCE HEADING
PRINT SOURCE DATA
100


26
PRINT SELECT ?
OFF
100
ON
DATARD
READ DATA RECORD FROM FILE 20
CREATE SOURCE HEADING
PRINT SOURCE DATA
100

NAME: PHASE4

TYPE: SUBROUTINE

GENERAL PURPOSE:

Prints input data, transfer functions and node voltages for the circuit analysis. Controls frequency sweeping. Tabulates plot data.

VARIABLES

AMPL = Amplitude values

F = Sum of frequency values

FNC = User-assigned frequency numbers

FREQ = Frequency values

FRTBL = Frequency number table

Subscript = Sequential frequency number

Value = User-assigned frequency number

GFREQ = Frequency values for single generator

ICOMP = Linear and nonlinear component type

IFP = User-assigned number of frequency to be plotted

IMPC = Impedance value

IMTPC = Impedance code

INOD = Nodes to be plotted

IORD = Orders to be plotted

IOUT = Logical unit number of line printer

JADD = Address of frequency data for sweep restoration

JRECO = Intermediate storage for driver file record number

MFREQ = Left-justified alphanumeric representation of user-assigned frequency number

ICCT = File 20 address of T.F. vector

ISTOFF = Print select switch. 1 = Suppress printing
0 = Enable printing

IT = Length of each T.F. vector in words

MAXIT = Total number of frequency sweep iterations

MXORD = Maximum order of analysis

NCOMP = Length of linear/nonlinear component data area

NPRT = Number of nodes specified in print select

NT = Total number of defined frequencies in circuit

NFREQ = Number of frequencies defined for given generator

NIMPS = Number of impedances defined for given generator

NIT = Frequency sweep iteration counter

NODES = Nodes specified in print select

NORDR = Order of analysis

NPLOT = Number of plot specifications

NVAMPL = Amplitude table

NVANG = Angle of node voltage

NVEC = Number of T.F. values for ordinate of plot

NVFREQ = Frequency value table

NVMAG = Magnitude of node voltage

NVOLT = Node voltage (rectangular form)

ORDPR = Number of orders specified in print select

ORDRS = Orders specified in print select

PLSW = Plot switch: 0 = Off

1 = On

PRSW = Print switch: 0 = Off

1 = On

SCFACT = Scale factor for T.F. to N.V. calculation

SFC = Frequency combination vector

SFT = Frequency heading

T = Transfer function element

VECTOR = T.F. values for ordinate of plot

SUBROUTINES CALLED

CXMPY CXPOL

DATAFD DATAVR DRIVDD PHASE1 PHASE5

PRNTIM PRSET PTIME PSHIFT TFTCNV

TIMOUT

CALLING PROGRAMS

PHASE3

DESCRIPTION:

Subroutine PHASE4 is the primary routine of NCAP's output

phase. It prints the output from a circuit analysis, tabulates plot data for use in a subsequent phase, and controls frequency sweeping. These functions are performed under the control of a driver file scan.

The standard printed output consists of all input values for each circuit element, as well as all scaled nonlinear transfer functions and node voltages resulting from a circuit analysis. The printing of circuit element data is controlled by the print select ON/OFF switch, which is initialized ON at the beginning of PHASE4 and subsequently reset whenever a print select driver record is processed. When a circuit element record is encountered in the driver file scan, the print select switch is interrogated. If the switch is ON, the circuit element data is printed if the switch is OFF, the printing of element data is suppressed.

The End Circuit driver record initiates printing of the transfer functions and node voltages under the control of the print select NODE and ORDER specifications. If no print selection is specified in the input stream, the complete transfer function vector and the corresponding node voltages are printed, one node per line, for each frequency combination at which the circuit was analyzed. The transfer functions and node voltages are output in both rectangular and log polar form in the order they were calculated by Phase 3.

If node and/or order print selection is specified, a table of selected nodes and orders is created during the processing of the print select driver record. When the End Circuit is

encountered, only those transfer functions corresponding to the nodes and orders in the print select table are printed.

Subroutine PHASE4 begins with initialization of phase parameters. The frequency counter is cleared by NF=0, the print select ON/OFF switch is turned ON by LSTOFF=0, and the logical unit number for the printer is stored in IOUT. The driver record number is initialized to DRVREC=1 and the print select node and order counters, NDPR and ORDPR are cleared.

A driver record is read from file 21 by subroutine DRIVRD. If MISC(7)=1, the printing of data associated with the driver record is by-passed, and control returns to the driver file scan. Otherwise subroutine PHASE4 responds to the MODE of the record as follows.

MODE=1 Driver File Header

The total number of input frequencies for the circuit is set by MXORD=MISC(2) and the frequency table is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The frequency values are transferred from BUFF(1)-(10) to the decimal vector FREQ and the frequency numbers are transferred from BUFF(11)-(20) to the integer vector FRTBL. Control then returns to the driver file scan.

MODE=13 Start Circuit

The printer carriage is restored in preparation for output, the length of each transfer function vector in words is calculated as LT=2*MISC(4), and control returns to the driver file scan.

MODE=3 Generator

The generator data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The number of frequencies and impedances associated with the generator are set by NFREQ=MISC(1) and NIMPS=MISC(2). The number of frequency sweep iterations is defined as NIT=MISC(5) and the generator driver record number is saved in JRECO. The user-assigned frequency numbers are transferred from BUFF(31)-(40) to the integer vector FNO.

The positive and negative nodes of connection are printed from MISC(3) and MISC(4) and the frequency number, frequency value, and complex amplitude for each of the NFREQ input frequencies are printed. The frequency values and amplitudes are appended to the tables NVFREQ and NVAMP to be used in converting transfer functions to node voltages.

For each impedance in the data record, the impedance code is transferred from the data buffer to IMTPC, and the impedance code and complex admittance are printed. After all of the impedance data has been output, control returns to the driver file scan.

MODE=4 Linear Components

MODE=5 Nonlinear Components

If the print select switch is off (LSTOFF.NE.0), the printing of component data is by-passed and control returns to the driver file scan. Otherwise the component data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The length of the component data record is set by NCOMP=MISC(1). For each component in the data record, its type, nodes of connection, and value (or coefficient set) is printed. Control then returns to the driver file scan.

MODE=6 Vacuum Diode

MODE=7 Vacuum Triode

MODE=8 Vacuum Pentode

MODE=9 Biploar Junction Transistor

MODE=10 Semiconductor Diode

MODE=20 Junction Field Effect Transistor

MODE=25 Linear Dependent Source

MODE=26 Nonlinear Dependent Source

If the print select switch is off (LSTOFF.NE.0), the printing of the device data is by-passed and control returns to the driver file scan. Otherwise the device data record is read

from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The device data is printed according to a predetermined format and control returns to the driver file scan.

MODE=17 Print Select

The number of orders and nodes specified in the print select function are stored by ORDPR=MISC(1) and NDPR=MISC(2). The print select ON/OFF switch is set by LSTOFF=MISC(3) and the print select data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. The orders to be printed are transferred from BUFF(1)-(10) to the integer vector ORDRS and the nodes to be printed are transferred from BUFF(11)-(20) to the integer vector NODES. Control returns to the driver file scan.

MODE=14 End Circuit

The printing of transfer functions and node voltages is initiated by the End Circuit driver record. The transfer functions are read from file 20, converted to node voltages, and printed in the order they were calculated in Phase 3. The transfer functions are accessed by recreating every frequency combination at which the circuit was analyzed while maintaining the file 20 address of the corresponding transfer function vector.

The End Circuit processing begins by initializing the address of the transfer function vector to LOCT=MISC (2)-LT. The order of analysis is initialized to NORDR=Ø and the print and plot switches are turned off by PRSW=Ø and PLSW=Ø.

Subroutine PRSET is called to generate the next frequency combination and to form the alphanumeric frequency heading. If the last frequency combination has not been generated (NORDR.GT. Ø), subroutine TFTONV is called to calculate the scale factor used in converting transfer functions to node voltages. The address of the next transfer function vector is calculated by LOCT=LOCT+LT.

When plot functions are included in the NCAP input (NPLOT. GT.Ø), the plot switch is turned on (PLSW=1) if the order of analysis NORDR is among the orders to be plotted. The plot switch is subsequently turned off (PLSW=Ø) if the frequency number is not among the frequencies to be plotted.

When print select order specifications are included in the NCAP input (ORDPR.GT.Ø), the print switch is turned on (PRSW=1) if the order of analysis NORDR is among the orders to be printed.

The frequency heading which contains the order of analysis, frequency value, and frequency combination is printed. If the print and plot switches are both off, further processing for the present frequency combination is by-passed, and the next frequency combination is generated.

Otherwise for each transfer function in the vector I=1, LT, 2 the corresponding node number is calculated by K=I/2+1 and the complex vector element is read from file 2Ø by subroutine DATARD

and stored in T. The transfer function is converted to log polar form by subroutine CXPOL and stored in POLAR.

When print select node specifications are included in the NCAP input (NDPR.CT.0), the printing of the transfer function is by-passed if the node K is not among the nodes in the print select table. Otherwise the transfer function is converted to a node voltage and the node number, transfer function (in both rectangular and polar form), and node voltage are printed.

If the plot switch is on and K is among the nodes to be plotted, the transfer function is appended to the plot table at VECTOR(J) for J=1, NPLOT.

After every frequency combination has been generated and the printing of transfer functions is complete (NORDR.LT.0), control returns to the driver file scan.

MODE=16 End

The End driver record controls frequency sweeping. If frequency sweeping is not specified for the circuit (MAXIT.LT.0), program control passes to subroutine PHASE5. Otherwise the frequency sweep iteration count is incremented by NIT=NIT+1 and the generator driver record is read from file 21 by subroutine DRIVPD.

If the frequency sweep is complete (NIT.GT.MAXIT), the original frequency data is read from file 20 by subroutine DATAPD and restored to the generator data record by subroutine

DATAWR. The frequency sweep iteration count is reset by NIT=0 and program control passes to subroutine PHASE5.

If the frequency sweep is not complete (NIT.LE.MAXIT), the next sweep iteration is initialized as follows: the address of the generator data record is saved in JADD and the driver record for the next sweep iteration is read from file 21 by subroutine DRIVRD. The sweep data is read from file 20 by subroutine DATARD and written to the generator data record by subroutine DATAWR. A new circuit analysis is initiated by calling subroutine PHASE1.

For other values of MODE, subroutine PHASE4 returns to the driver file scan without further processing.

NAME: HHMMSS

TYPE: SUBROUTINE

GENERAL PURPOSE:

Converts clock time for NCAP timing routines to hours, minutes, and seconds.

VARIABLES:

ATIME = Intermediate storage for time calculation

HRS = Hours of clock time

MIN = Minutes of clock time

SEC = Seconds of clock time

TIME = Clock time

SUBROUTINES CALLED:

NONE

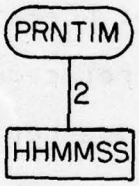
CALLING PROGRAMS:

PRNTIM

DESCRIPTION:

Subroutine HHMMSS converts clock time for the NCAP timing routines to hours, minutes, and seconds. The time, expressed in hours, tenths, and hundredths, is transmitted to the subroutine in the decimal argument TIME. The converted time is returned to the calling program in the integer arguments HRS, MIN, and SEC.

Hours of clock time are derived by taking the integer part of TIME. Minutes are calculated by multiplying the fractional part of TIME by 60., and retaining the integer part of the result. Seconds are then found by multiplying the hundredths portion of TIME by 60., and subroutine HHMMSS returns to the calling program.


PRNTIM
2
HHMMSS

NAME: PRNTIM

TYPE: SUBROUTINE

GENERAL PURPOSE:

Prints output from timing routines

VARIABLES:

ICUT = Logical unit number of line printer

LIST = Alphanumeric designator of program segements

SUM1 = Total clock time for one circuit

SUM2 = Total clock time for NCAP job

TIME = Elapsed clock time per segment

TTOTAL = Accumulated clock time per segment

SUBROUTINES CALLED:

UMMSS

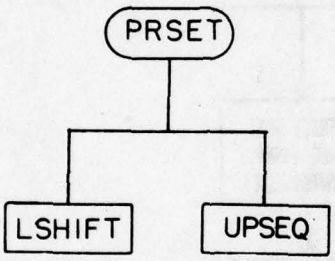
CALLING PROGRAMS:

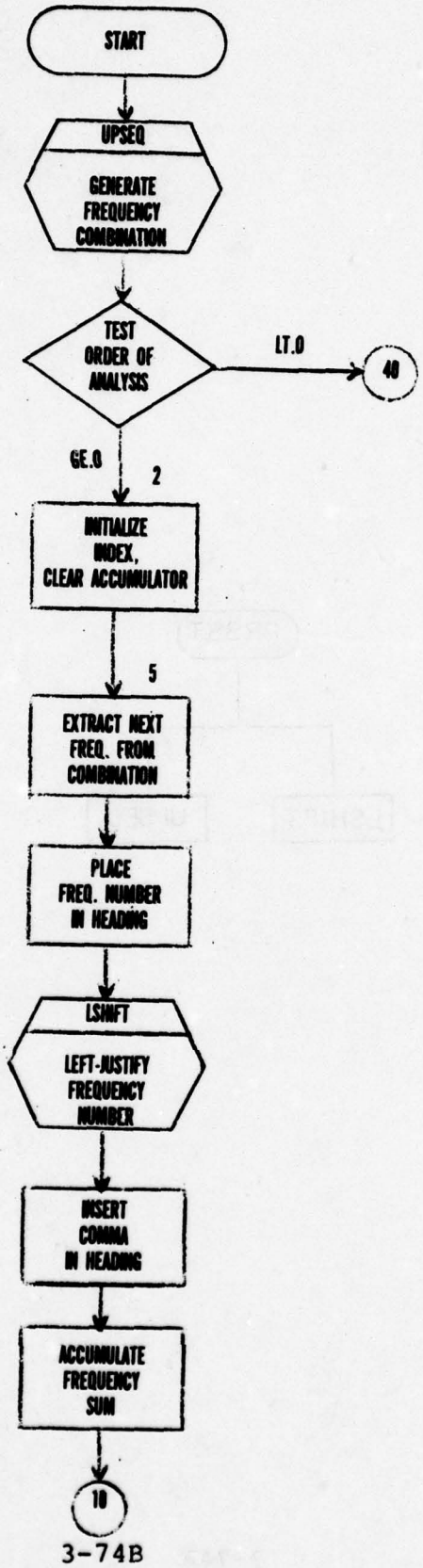
PHASE4

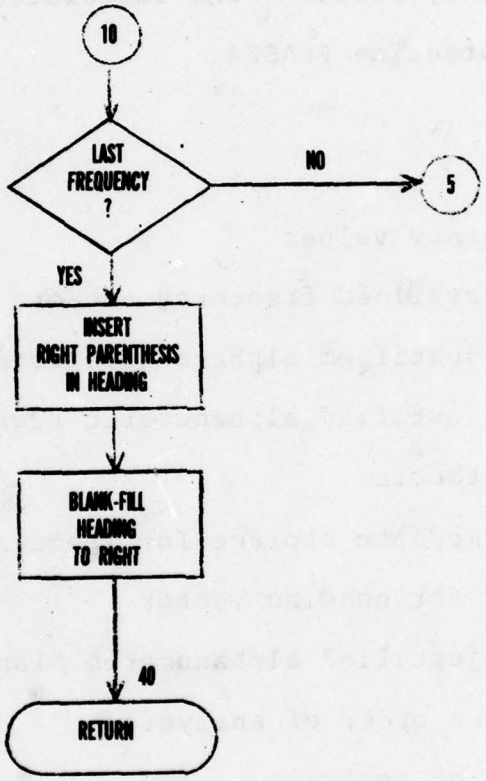
DESCRIPTION:

Subroutine PRNTIM prints the output from the NCAP timing routines. Elapsed and accumulated times for each program segment, TIME and TTOTAL, are transmitted to the subroutine in the labelled common area TIMEL.

First the total time for the past circuit analysis and total time for the NCAP job are accumulated in SUM1 and SUM2. Then for each program segment executed during the previous circuit analysis (TIME(I).GT.0), the elapsed and accumulated segment times are converted to hours, minutes, and seconds by subroutine HHMMSS and printed in tabular form. After each segment's time has been printed, the TIME array is zeroed and subroutine PRNTIM returns to the calling program.


PRSET
LSHIFT
UPSEQ


START
UPSEQ
GENERATE FREQUENCY COMBINATION
TEST ORDER OF ANALYSIS
LT.0
40
GE.0
2
INITIALIZE INDEX, CLEAR ACCUMULATOR
5
EXTRACT NEXT FREQ. FROM COMBINATION
PLACE FREQ. NUMBER IN HEADING
LSHIFT
LEFT-JUSTIFY FREQUENCY NUMBER
INSERT COMMA IN HEADING
ACCUMULATE FREQUENCY SUM
10


10
LAST FREQUENCY ?
NO
5
YES
INSERT RIGHT PARENTHESIS IN HEADING
BLANK-FILL HEADING TO RIGHT
40
RETURN

NAME: PRSET

TYPE: SUBROUTINE

GENERAL PURPOSE:

Controls order of analysis, forms frequency combinations, creates frequency heading, and calculates sum of frequencies for use by subroutine PHASE4.

VARIABLES:

FREQ = Frequency values

FRTBL = User-assigned frequency numbers

ICMMA = Left-justified alphanumeric comma

IRPTR = Left-justified alphanumeric right parenthesis

ISFQ = Intermediate storage for frequency number

L = Index for heading vector

IBLANK = Left-justified alphanumeric blank

MXORD = Maximum order of analysis

NORDR = Order of analysis

NUM = Index for frequency number and value

S = Sum of frequencies

SFQ = Frequency combination vector

SET = Alphanumeric frequency heading

SUBROUTINES CALLED

ISHIFT UPSFQ

CALLING PROGRAMS:

PHASE4

DESCRIPTION:

Subroutine PRSET forms the frequency combinations and creates the alphanumeric frequency heading for use by subroutine PHASE4. Arguments transmitted to the subroutine from the calling program are: FREQ, the circuit's input frequency values; FRTBL, the frequency number table; MXORD, the number of input frequencies for the circuit (maximum order of analysis); NORDR, the number of frequencies in the last frequency combination (order of analysis); and SEQ, the previous frequency combination vector. Arguments returned to subroutine PHASE4 are: NORDR, the number of frequencies in the new frequency combination (updated order of analysis); S, the calculated frequency sum for the new frequency combination; SET, the alphanumeric frequency heading; and SEQ, the updated frequency combination vector.

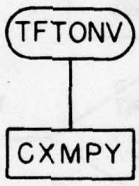
Upon entering subroutine PRSET, the next frequency combination is generated by subroutine UPSEQ. If the last possible frequency combination has already been generated (NORDR.LE.0), subroutine PRSET returns to the calling program without further processing. Otherwise NORDR represents the number of frequencies in the new combination, which is stored according to increasing sequential frequency numbers in the integer vector SEQ.

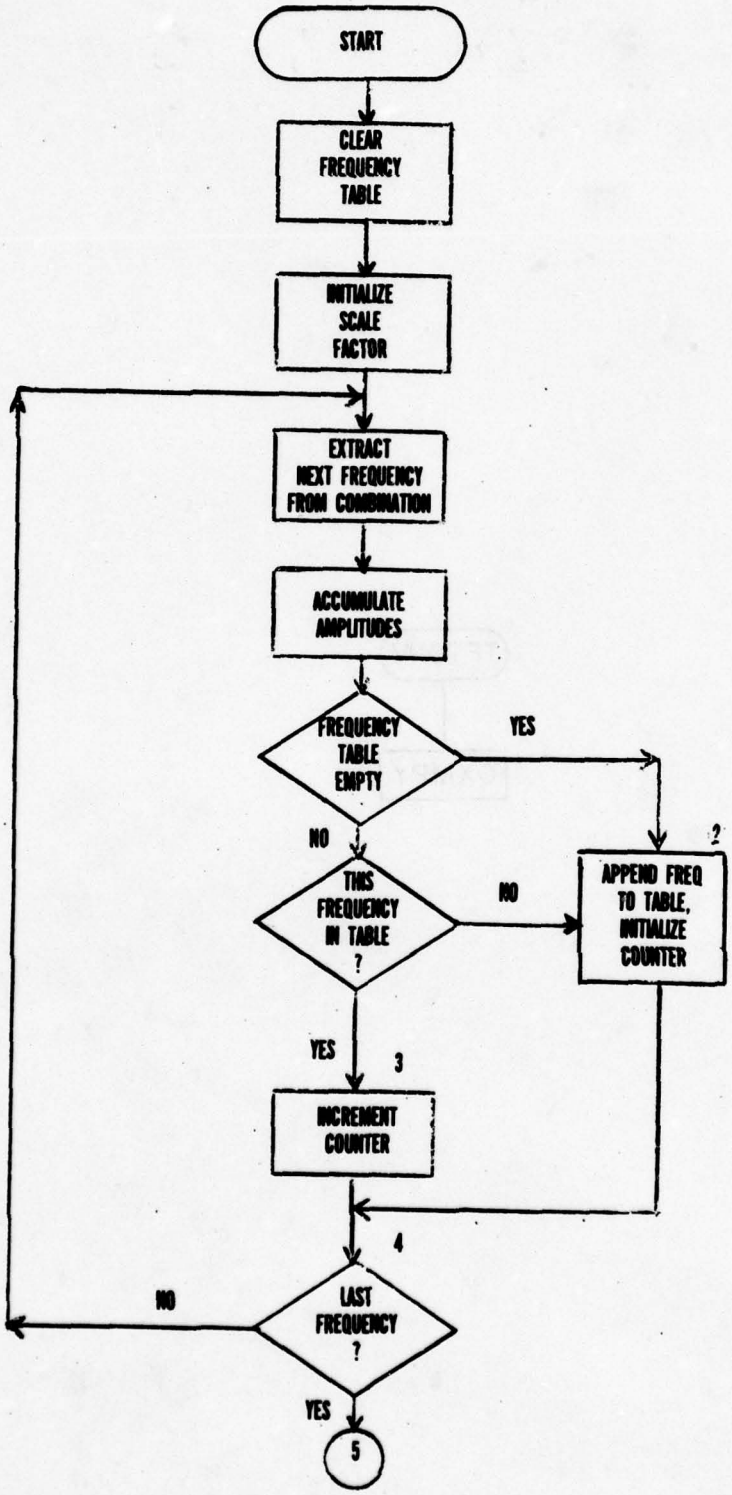
The frequency sum is initialized to S=Ø. and the index for the frequency heading is set at L=1. The frequency heading is created by converting each sequential frequency number in the frequency combination to its user-assigned number. The user-assigned number is then converted to left-justified alphanumeric format and placed in the heading.

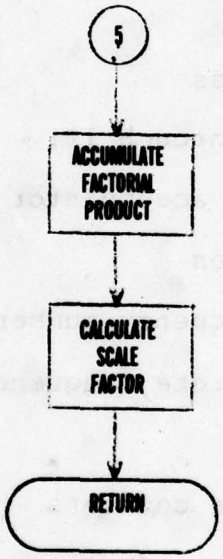
First the frequency combination vector is decoded as follows: the vector element SEQ(I) is stored in ISEQ. Then ISEQ is successively shifted right one bit at a time while a count, NUM, is maintained on the number of shifts performed. When ISEQ=1, then NUM contains the sequential frequency number represented by SEQ(I).

The corresponding user-assigned frequency number is taken from the frequency table as FRTBL(NUM), stored in the frequency heading at SET(L), and converted to left-justified alphanumeric format by subroutine LSHIFT. A comma is inserted after the frequency number in the heading by SET(L+1)=ICMMA and the heading index is updated to L=L+2. Then the frequency value FREQ(NUM) is added to the frequency sum.

After every frequency in the combination has been placed in the heading, a right parenthesis is inserted after the last frequency number by SET(L-1)=IRPTR, the heading is blank-filled to the right, and subroutine PPSET returns to the calling program.


TFTONV
CXMPY


START
CLEAR FREQUENCY TABLE
INITIALIZE SCALE FACTOR
EXTRACT NEXT FREQUENCY FROM COMBINATION
ACCUMULATE AMPLITUDES
FREQUENCY TABLE EMPTY
YES
NO
2
THIS FREQUENCY IN TABLE ?
NO
APPEND FREQ TO TABLE, INITIALIZE COUNTER
YES
3
INCREMENT COUNTER
4
LAST FREQUENCY ?
NO
YES
5


5
ACCUMULATE FACTORIAL PRODUCT
CALCULATE SCALE FACTOR
RETURN

NAME: TFTONV

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates scale factor for use in converting transfer functions to node voltages

VARIABLES:

AMPL = Amplitude values

FCTRL = Values of 1! through 10!

FPROD = Multiplicative accumulator

CFREQ = Frequency values

INDEX = Sequential frequency number

IPTR = Number of discrete frequency values in combination

M = Frequency value counters

MVALUE = Table of discrete frequency values in combination

NORDR = Order of analysis

SCFACT = Calculated scale factor

SEQ = Frequency combination vector

TEMP = Intermediate calculation

SUBROUTINES CALLED:

CXMPY

CALLING PROGRAMS:

PHASE4

DESCRIPTION:

Subroutine TFTCNV calculates the scale factor for use in converting transfer functions to node voltages for use by subroutine PHASE4. The arguments transmitted to subroutine TFTCNV from the calling program are: NORDR, the number of frequencies in the frequency combination (order of analysis) GFREQ, the circuit's input frequency values; AMPL, the circuit's input amplitude values; and SEQ, the frequency combination vector. The calculated scale factor is returned to the calling program in SCFACT.

For a given frequency combination, the scale factor is a function of the amplitude at each frequency in that combination, the order of analysis, and the number of times each discrete frequency value appears in the combination. Subroutine TFTCNV creates a table of the discrete frequency values in the combination and counts the number of times each value is used in the frequency combination. The discrete frequency values and the corresponding counters are stored in the vectors MVALUE and M so that M(I) counts the number of times MVALUE(I) appears in the combination. The length of the MVALUE and M vectors is accumulated in IPTR.

Subroutine TFTCNV begins by clearing the MVALUE and M vectors and setting IPTR=0. The complex scale factor is

initialized to SCFACT=(1., 0.). Then for each element in the frequency combination vector, SEQ(1) through SEQ(NORDR), the sequential frequency number represented by SEQ(I) is decoded by the function IND and stored in INDEX. Subroutine CXMPY is called to multiply the accumulated scale factor by the amplitude AMP(INDEX), storing the product in SCFACT.

If no frequency values have been placed in the table (IPTR =0), the table length is updated to IPTR=IPTR+1, the value CFREQ(INDEX) is appended to the table at MVALUE(IPTR), and the frequency counter M(IPTR) is initialized to 1.

If there are frequency values in the table (IPTR.GT.0), the value CFREQ(INDEX) is compared to each MVALUE. If no match is found the new value is appended to the table as above. If a match is found, the corresponding counter M(J) is incremented.

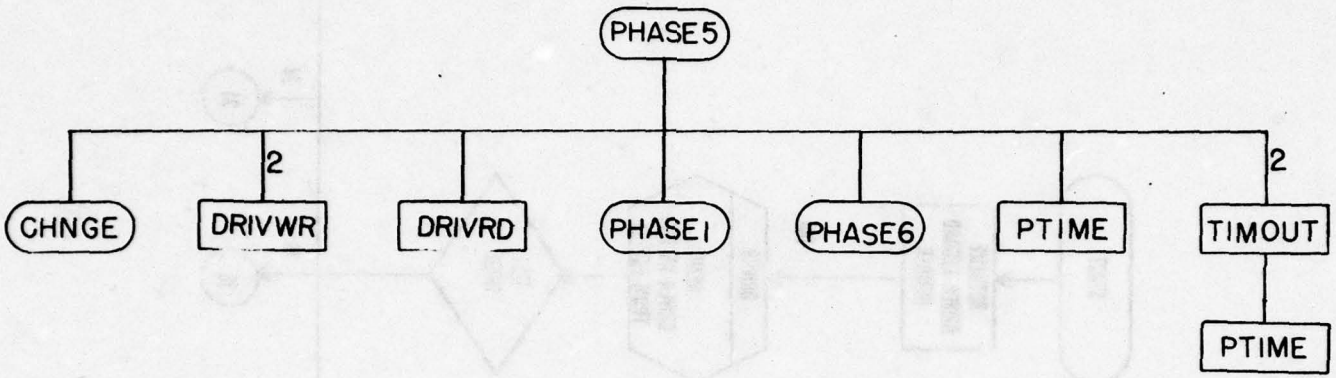
After all NORDR frequencies in the SEQ vector have been processed, the values of M(I)! for every frequency counter are multiplied together and stored in FPROD. After the intermediate calculation:

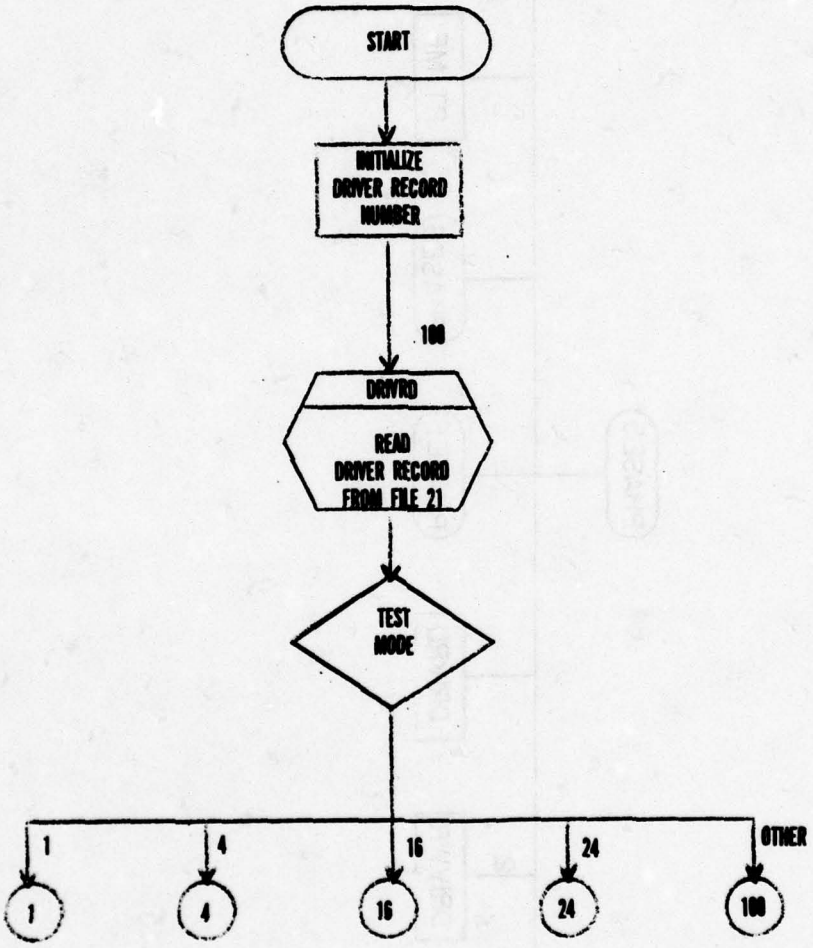
$$TEMP=NORDR! \; / \; FPROD*2^{NORDR-1}$$

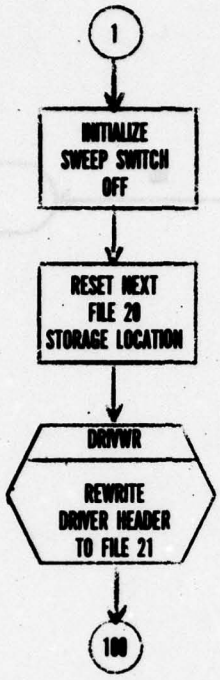
has been performed, the complex scale factor follows directly by multiplying the real and imaginary parts of SCFACT by TEMP.

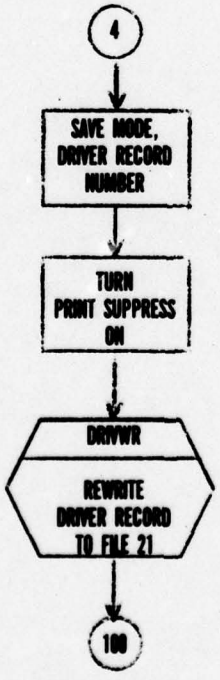
PHASE5

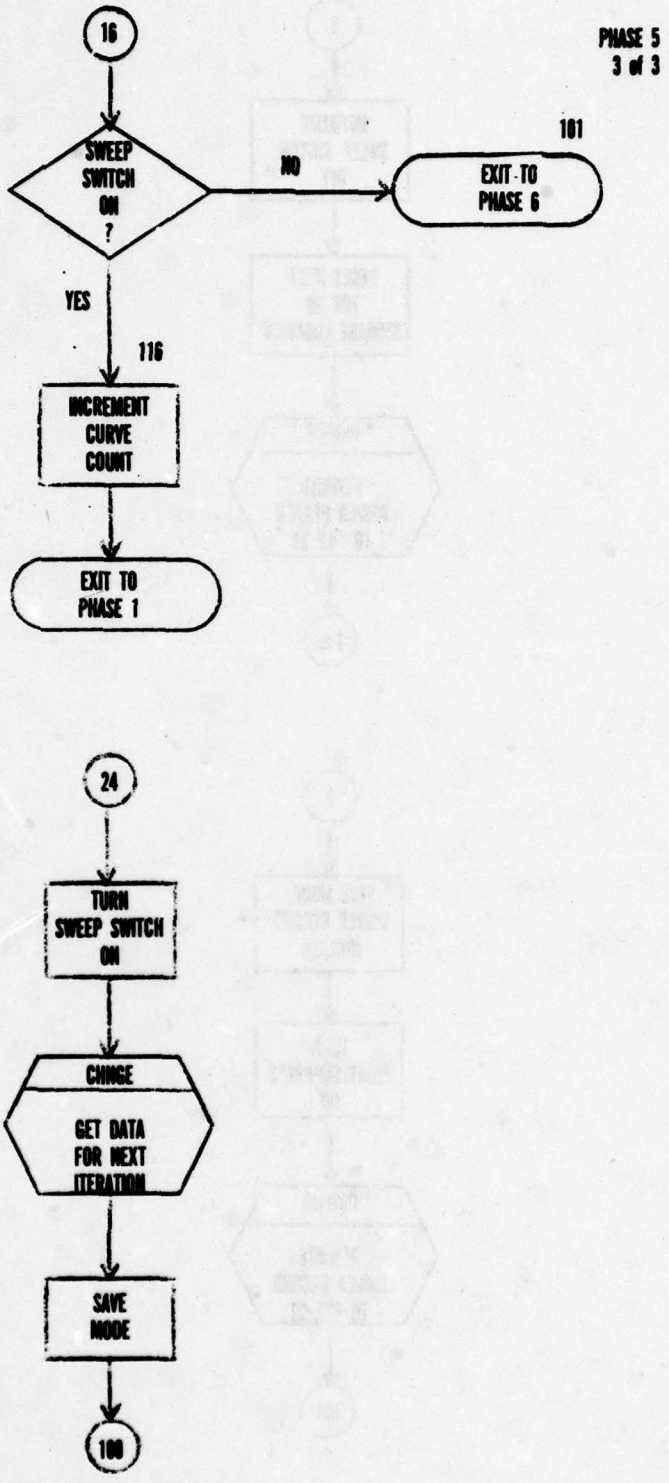
CHNGE


PHASE5
2
2
CHNGE
DRIVWR
DRIVRD
PHASE1
PHASE6
PTIME
TIMOUT
PTIME


START
INITIALIZE DRIVER RECORD NUMBER
100
DRIVRD
READ DRIVER RECORD FROM FILE 21
TEST MODE
1
4
16
24
OTHER
1
4
16
24
100


1
INITIALIZE SWEEP SWITCH OFF
RESET NEXT FILE 20 STORAGE LOCATION
DRVWR
REWRITE DRIVER HEADER TO FILE 21
100


4
SAVE MODE, DRIVER RECORD NUMBER
TURN PRINT SUPPRESS ON
DRVWR
REWRITE DRIVER RECORD TO FILE 21
100


16
PHASE 5
3 of 3
SWEEP SWITCH ON ?
NO
101
EXIT TO PHASE 6
YES
116
INCREMENT CURVE COUNT
EXIT TO PHASE 1
24
TURN SWEEP SWITCH ON
CHNGE
GET DATA FOR NEXT ITERATION
SAVE MODE
100

NAME: PHASE 5

TYPE: SUBROUTINE

GENERAL PURPOSE:

COntrols linear component sweeping

VARIABLES:

IDONE = Sweep switch: 0 = No sweep

1 = Sweep

JRECC = Intermediate storage for driver file record number

MDESV = Intermediate storage for device mode

NCURV = Number of curves defined by circuit analysis

SUBROUTINES CALLED:

CHNGE DRIVRD DRIVWR PHASE1 PHASE6

PTIME TIMOUT

CALLING PROGRAMS:

PHASE4

DESCRIPTION:

Subroutine PHASE5 controls linear component sweeping by initiating a new circuit analysis for each iteration of a component sweep in the circuit description. The method is to

scan the driver file for sweep functions which are represented by a MODE=4 (linear component) driver record followed by a MODE= 24 (linear component sweep) driver record. For each such function, the component values for the next iteration are brought forward from the sweep data record to replace the previous linear component data, and a new circuit analysis is performed beginning with Phase 1.

The driver file scan begins by initializing the driver file record number to DRVREC=1. A driver record is read from file 21 by subroutine DRIVRD, and subroutine PHASE5 responds to the MODE of the record as follows:

MODE=1 Driver File Header

The sweep switch is turned off by IDONE=0, indicating that no sweep functions have been processed. The file 20 storage address is reset by MISC(1)=MISC(4), allowing the admittance matrices and transfer function vectors for the next circuit analysis to be written over those from the last analysis . The updated driver file header is rewritten by subroutine DRIVWR and control returns to the driver file scan.

MODE=4 Linear Components

The device mode and driver record number are saved by

MDESV=4 and JRECO=DRVREC-1 in preparation for a possible sweep. MISC(7) is set to 1 to suppress printing of the old component values in PHASE4. The updated linear component driver record is rewritten by subroutine DRIVWR and control returns to the driver file scan.

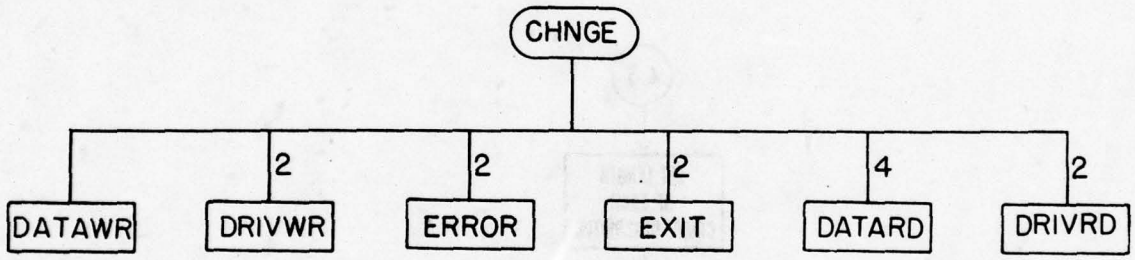
MODE=24 Linear Element Sweep

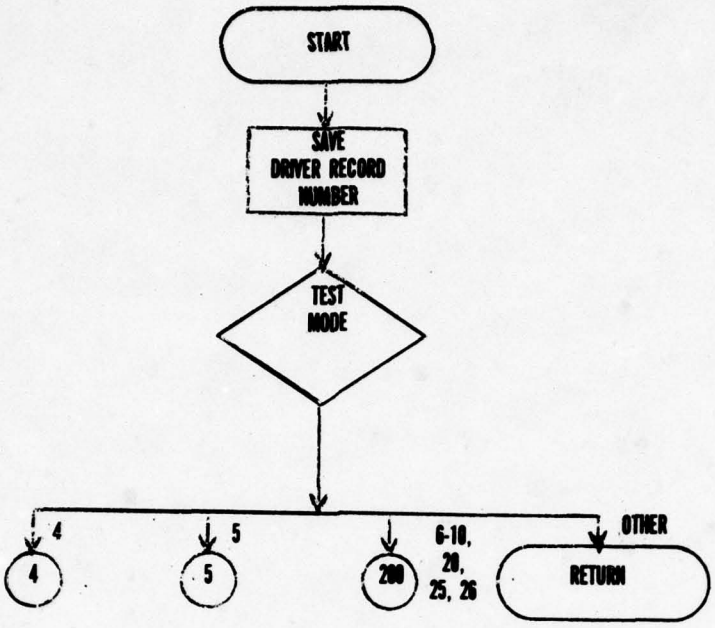
The sweep switch is turned on by IDONE=1 and subroutine CHNGE is called to replace the previous component data with the data values for the new sweep iteration. The saved device mode is changed to MDESV=24, causing any remaining sweep records associated with the present linear component definition to be by-passed in the driver scan (see subroutine CHNCF), and control returns to the driver file scan.

MODE=16 End

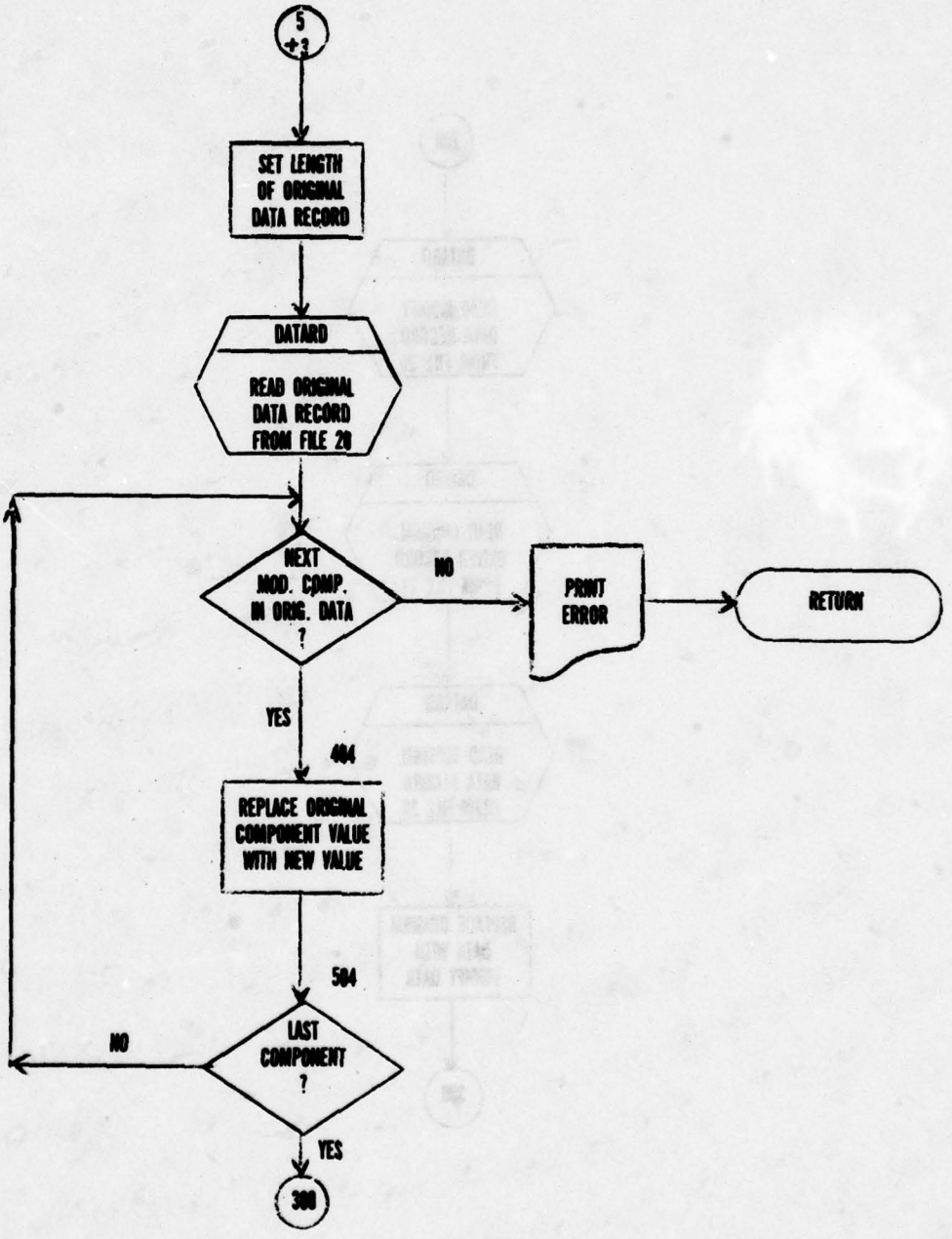
If the sweep switch is on (IDONE=1), the curve count is incremented by NCURV=NCURV+1 and a new circuit analysis is initiated by calling subroutine PHASE1. If the sweep switch is off (IDONE=0), program control passes to subroutine PHASE6.

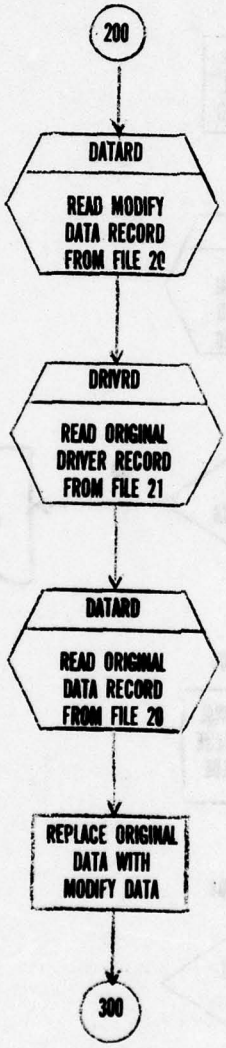
For other values of MODE, control returns to the driver file scan without further processing by subroutine PHASE5.


CHNGE
2
2
2
4
2
DATAWR
DRIVWR
ERROR
EXIT
DATARD
DRIVRD


START
SAVE DRIVER RECORD NUMBER
TEST MODE
4
5
6-10, 20, 25, 26
OTHER
4
5
200
RETURN


4, 5
SET LENGTH OF EACH COMP. DESCRIPTION
704
DATARD
READ NEW DATA RECORD FROM FILE 20
SET LENGTH OF NEW DATA RECORD
DRIVRD
READ ORIGINAL DRIVER RECORD FROM FILE 21
5 +3


5
+3
SET LENGTH OF ORIGINAL DATA RECORD
DATARD
READ ORIGINAL DATA RECORD FROM FILE 20
NEXT MOD. COMP. IN ORIG. DATA ?
NO
PRINT ERROR
RETURN
YES
404
REPLACE ORIGINAL COMPONENT VALUE WITH NEW VALUE
504
LAST COMPONENT ?
NO
YES
300


200
DATARD
READ MODIFY
DATA RECORD
FROM FILE 20
DRIVRD
READ ORIGINAL
DRIVER RECORD
FROM FILE 21
DATARD
READ ORIGINAL
DATA RECORD
FROM FILE 20
REPLACE ORIGINAL
DATA WITH
MODIFY DATA
300


300
TURN PRINT SUPPRESS SWITCH OFF
DRIVWR
REWRITE ORIGINAL DRIVER RECORD TO FILE 21
DATAWR
WRITE UPDATED DEVICE DATA TO FILE 20
SET MODE OF MODIFY RECORD TO NO-OP
DRIVWR
REWRITE MODIFY DRIVER RECORD TO FILE 21
RETURN

NAME: CHNGE

TYPE: SUBROUTINE

GENERAL PURPOSE:

Performs data replacement for linear component sweep and device modification

VARIABLES:

BUFF = I/O buffer for modified data

BUFFO = I/O buffer for original data

JRCSV = Address of modify driver record

JRFCO = Address of driver record of device being modified

MDESV = Mode of device being modified

NCMP = Length of modified component data area

NCMPO = Length of original component data area

NW = Length of each component description

SUBROUTINES CALLED

DATARD DATAWR DRIVRD DRIVWR ERROR

CALLING PROGRAMS:

PHASE5 PHASE6

DESCRIPTION:

Subroutine CHNGE performs the data replacement for linear component sweeping and device modification performed in Phases 5 and 6. Arguments transmitted to the subroutine are: MDESV, the mode of the device being modified and JRECO, the driver record number of the device being modified. The modify or sweep driver record is transmitted to subroutine CHNGE in the first ten words of global common.

First the driver record number of the sweep or modify record from which the data is derived is set by JRCSV=DRVREC-1. Then subroutine CHNGE interrogates the mode of the device being modified, responding to the value of MDESV as follows:

MDESV=4 Linear Components

MDESV=5 Nonlinear Components

The length of each component description in the data record is set to NW=4 for linear components or NW=13 for nonlinear components. The sweep or modify data record is read from file 20 by subroutine DATARD and stored in the common data buffer BUFF. Its length is saved by NCMP=MISC(1).

The driver record number is reset to DRVREC=JRECO and the original component driver record is read by subroutine DRIVRD. Then the component data record is read from file 20 by subroutine DATARD and stored in BUFFO. Its length is saved by NCMPO=MISC(1).

The type and nodes of connection for each component in the modify data record (BUFF(I), BUFF(I+1), and BUFF(I+2) for I=1, NCMP,NW) are compared against those in the original component data record (BUFFO(J), BUFFO(J+1) and BUFFO(J+2) for J=1, NCMPO, NW). If no match is found, an appropriate error message is printed and the program terminates. When a match is found, the original component value or coefficients are replaced with the modified component value or coefficients according to:

BUFFO(J-1+K)=BUFF(I-1+K) for K=4, NW

The component driver record is updated by MISC(7)=0 to enable printing of the new component data in PHASE4. The updated component driver record is rewritten by subroutine DRIVWR and the updated component data record is written to file 20 by subroutine DATAWR. The sweep or modify driver record is disabled by writing a no-operation (MODE=2) record in its place in the driver file and subroutine CHNGE returns to the calling program.

MDESV=6 Vacuum Diode

MDESV=7 Vacuum Triode

MDESV=8 Vacuum Pentode

MDESV=9 Bipolar Junction Transistor

MDESV=10 Semiconductor Diode

MDESV=20 Junction Field Effect Transistor

MDESV=25 Linear Dependent Source

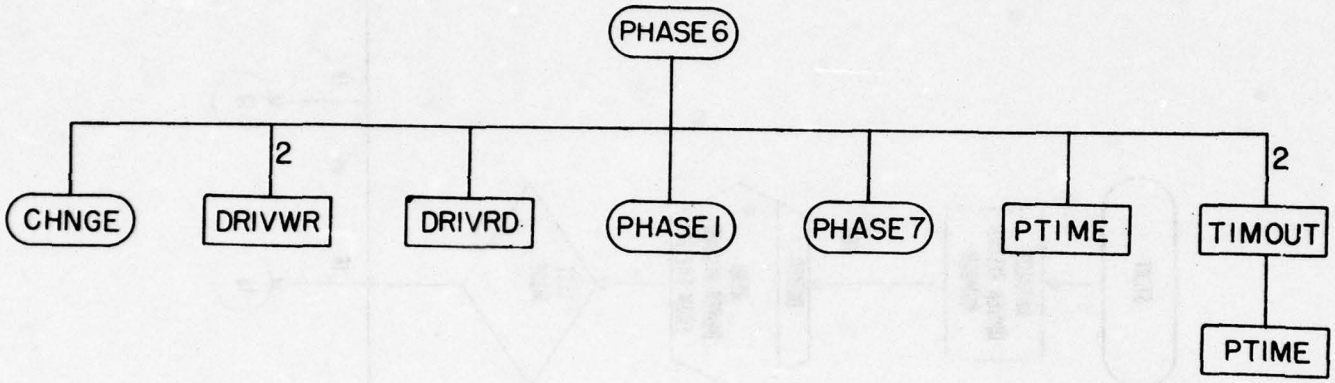
MDESV=26 Nonlinear Dependent Source

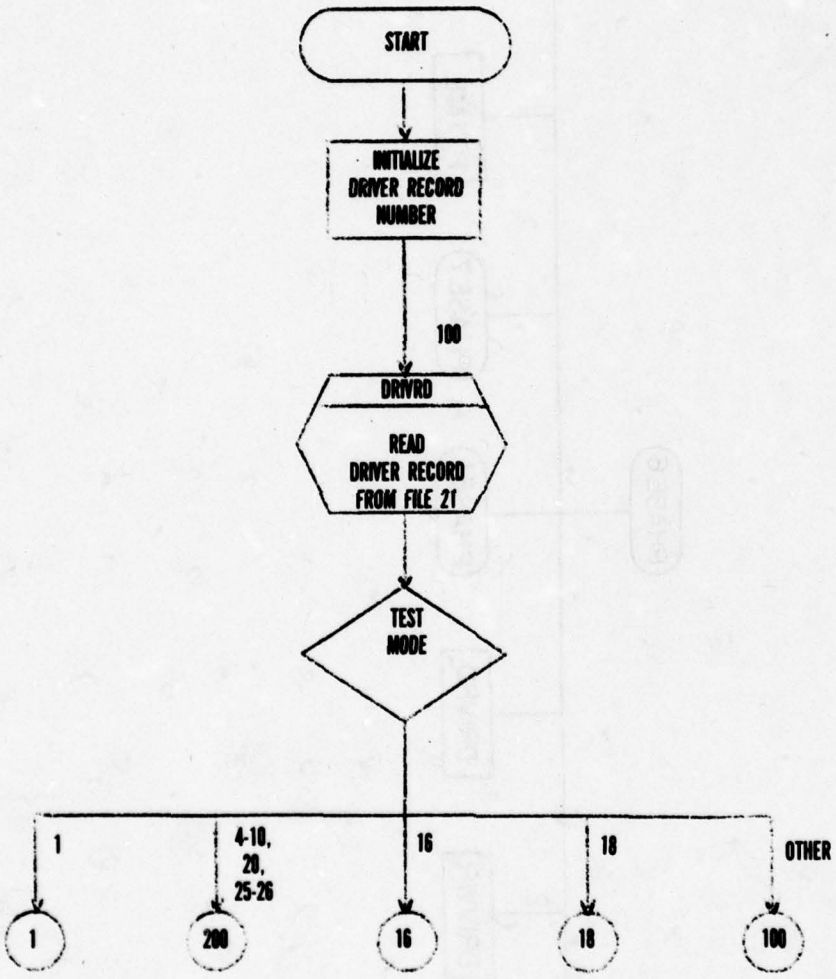
The modify data record is read from file 20 by subroutine DATARD and stored in BUFFO. The driver record number is reset by DRVREC=JRECO and the original device driver record is read from file 20 by subroutine DRIVRD.

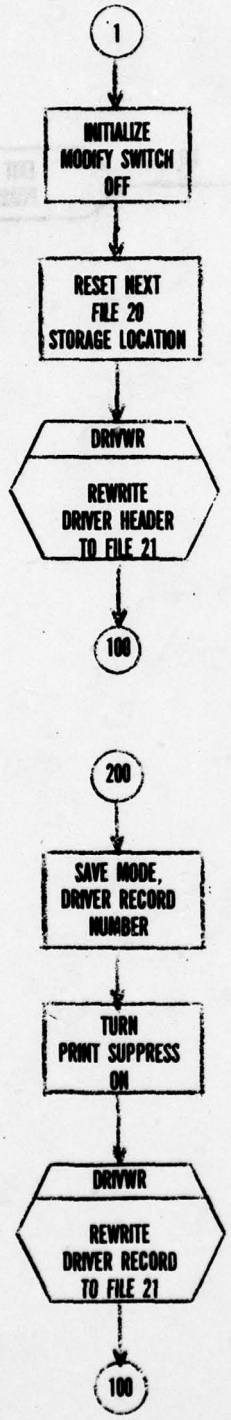
The component driver record is them updated by MISC(7)=0 to enable the printing of the new device data in PHASE4. The updated device driver record is rewritten by subroutine DRIVWR and the modify data record is written from BUFFO to the original device data record on file 20 by subroutine DATAWR. The modify driver record is disabled by writing a no-operation (MODE=2) record in its place in the driver file and subroutine CHNGE returns to the calling program.

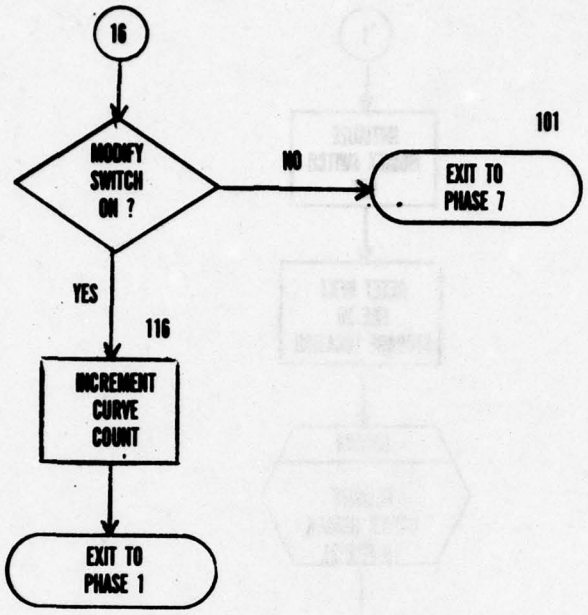
Any other value of MDESV encountered by subroutine CHNGE indicates a misplaced modify function. An appropriate error message is printed and the program terminates.

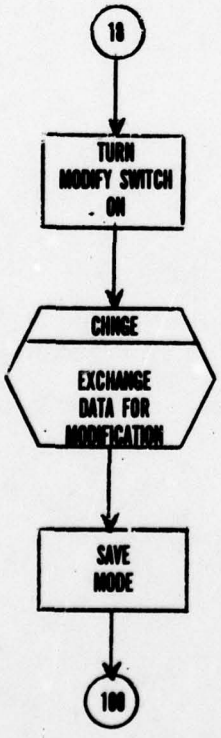
PHASE6


PHASE6
CHNGE
2
DRIVWR
DRIVRD
PHASE1
PHASE7
PTIME
2
TIMOUT
PTIME


START
INITIALIZE DRIVER RECORD NUMBER
100
DRIVRD
READ DRIVER RECORD FROM FILE 21
TEST MODE
1
4-10, 20, 25-26
16
18
OTHER
1
200
16
18
100


1
INITIALIZE
MODIFY SWITCH
OFF
RESET NEXT
FILE 20
STORAGE LOCATION
DRIVWR
REWRITE
DRIVER HEADER
TO FILE 21
100
200
SAVE MODE,
DRIVER RECORD
NUMBER
TURN
PRINT SUPPRESS
ON
DRIVWR
REWRITE
DRIVER RECORD
TO FILE 21
100


16
MODIFY SWITCH ON ?
NO
101
EXIT TO PHASE 7
YES
116
INCREMENT CURVE COUNT
EXIT TO PHASE 1


18
TURN MODIFY SWITCH ON
CHNGE
EXCHANGE DATA FOR MODIFICATION
SAVE MODE
100

NAME: PHASE6

TYPE: SUBROUTINE

GENERAL PURPOSE:

Controls device modification

VARIABLES

IDCNF = Modify switch: 0 = No modify

1 = Modify

JDECC = Intermediate storage for driver file record number

MDESV = Intermediate storage for device mode

NCURV = Number of curves defined by circuit analysis

SUBROUTINES CALLED:

CDFCF DRIVED DRIVWP PHASE1 PHASE7

RTIME TIMOUT

CALLING PROGRAMS

PHASE5

DESCRIPTION:

Subroutine PHASE6 controls linear and nonlinear element modifications (except for generator modifications). It scans the driver file for modify functions which are represented by a

device driver record (MODE=4,5,6,7,8,9,10,20,25,26) followed by a modify record (MODE=18). For each such function, the modify data record is brought forward to replace the original device data, and a new circuit analysis is performed beginning with Phase 1.

The driver file scan begins by initializing the driver file record number to DFVEC=1. A driver record is read from file 21 by subroutine DRIVRD, and subroutine PHASE6 responds to the MODE of the record as follows:

MODE=1 Driver File Header

The modify switch is turned off by IDONE=0, indicating that no modify functions have been processed. The file 20 storage address is reset by MISC(1)=MISC(4), allowing the admittance matrices and transfer function vectors for the next circuit analysis to be written over those from the last analysis. The updated driver file header is rewritten by subroutine DRIVWR and control returns to the driver file scan.

MODE=4 Linear Components

MODE=5 Nonlinear Components

MODE=6 Vacuum Diode

MODE=8 Vacuum Pentode

MODE=9 Bipolar Junction Transistor

MODE=10 Semiconductor Diode

MODE=20 Junction Field Effect Transistor

MODE=25 Linear Dependent Source

MODE=26 Nonlinear Dependent Source

The device mode and driver record number are saved by MDESV=MODE and JRECO=DRVREC-1 in preparation for a possible modification. MISC(7) is set to 1 to suppress printing of the old device values in PHASE4. The updated device driver record is rewritten by subroutine DRIVER and control returns to the driver file scan.

MODE=18 Modify

The modify switch is turned on by IDONE=1 and subroutine CHNGE is called to replace the original device data with the modify data. The saved device mode is changed to MDESV=18, causing any remaining modify records associated with the present device to be by-passed in the driver scan (see subroutine CHNGE), and control returns to the driver file scan.

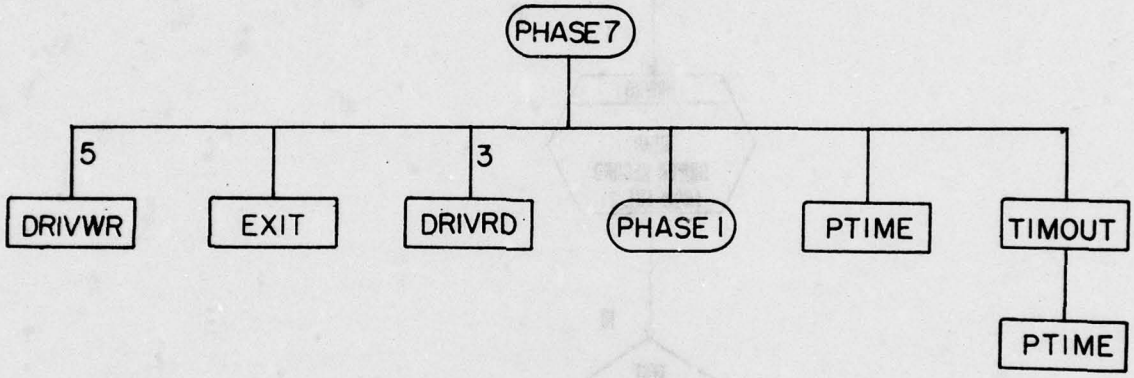
MODE=16 End

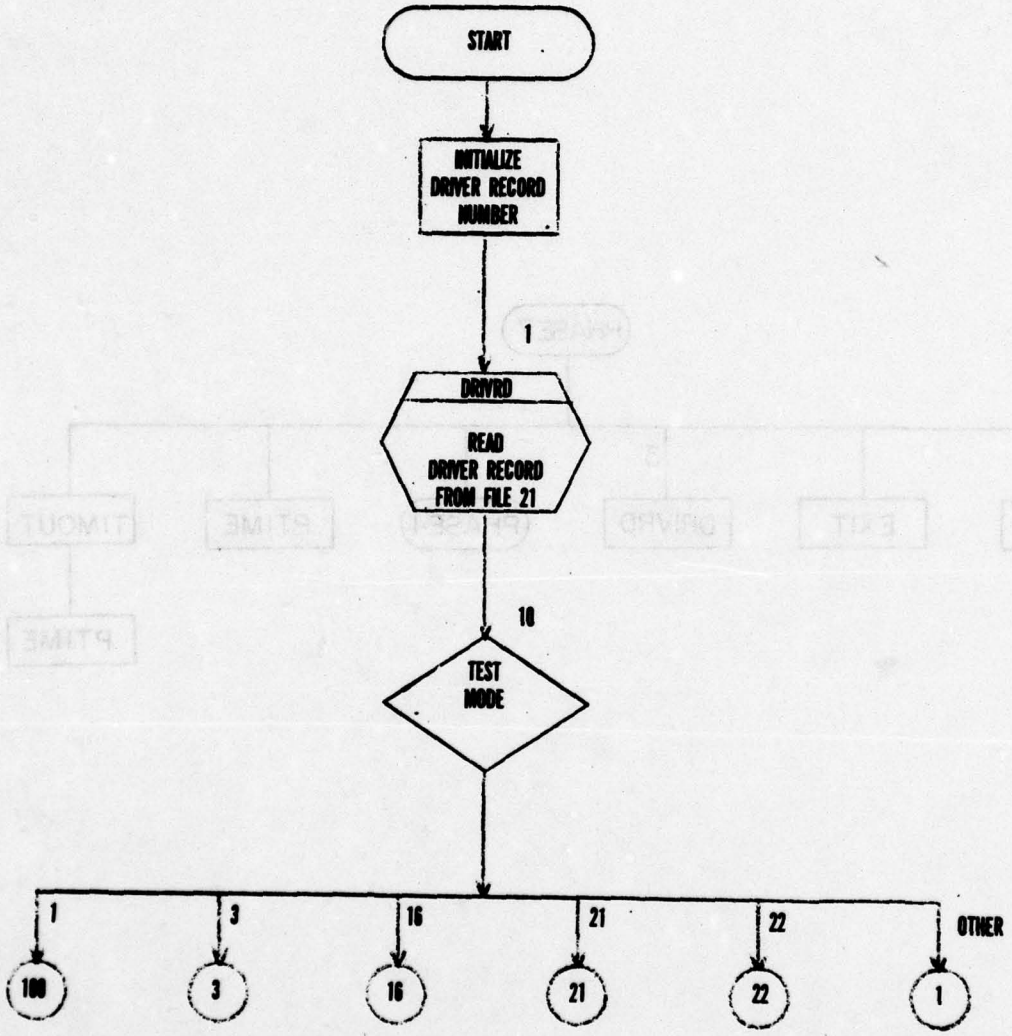
If the modify switch is on (IDONE=1), the curve count is incremented by NCURV=NCURV+1 and a new circuit analysis is initiated by calling subroutine PHASE1. If the modify switch is

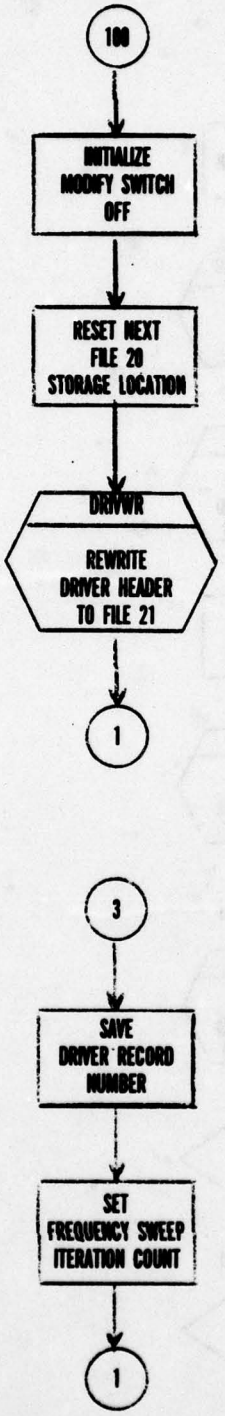
off (IDCNE=0), program control passes to subroutine PHASE7.

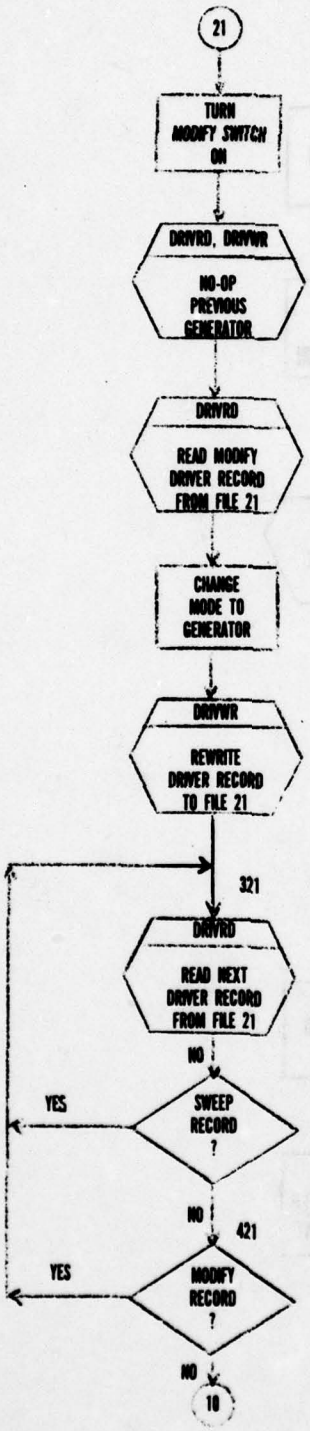
For other values of MODE, control returns to the driver file scan without further processing by subroutine PHASE6.

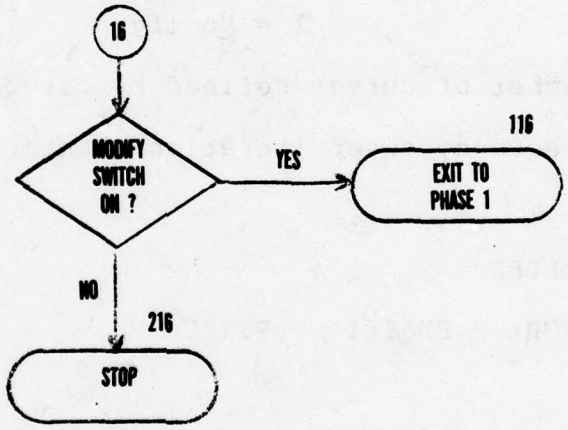
PHASE7


PHASE7
5
3
DRIVWR
EXIT
DRIVRD
PHASE1
PTIME
TIMOUT
PTIME


START
INITIALIZE DRIVER RECORD NUMBER
1
DRIVRD
READ DRIVER RECORD FROM FILE 21
10
TEST MODE
1
3
16
21
22
OTHER
100
3
16
21
22
1


100
INITIALIZE
MODIFY SWITCH
OFF
RESET NEXT
FILE 20
STORAGE LOCATION
DRIVWR
REWRITE
DRIVER HEADER
TO FILE 21
1
3
SAVE
DRIVER RECORD
NUMBER
SET
FREQUENCY SWEEP
ITERATION COUNT
1


21
TURN MODIFY SWITCH ON
DRVRD, DRVWR
NO-OP PREVIOUS GENERATOR
DRVRD
READ MODIFY DRIVER RECORD FROM FILE 21
CHANGE MODE TO GENERATOR
DRVWR
REWRITE DRIVER RECORD TO FILE 21
321
DRVRD
READ NEXT DRIVER RECORD FROM FILE 21
NO
YES
SWEEP RECORD ?
NO
421
YES
MODIFY RECORD ?
NO
10


16
MODIFY SWITCH ON ?
YES
116
EXIT TO PHASE 1
NO
216
STOP

NAME: PHASE7

TYPE: SUBROUTINE

GENERAL PURPOSE:

Controls generator modification

VARIABLES

JPECO = Address of generator driver record

MODFY = Modify switch; 0 = No modify

1 = Modify

NCURV = Number of curves defined by circuit analysis

NIT = Frequency sweep iteration counter

SUBROUTINES CALLED

DRIVRD DRIVWR PHASE1 PTIME

TIMOUT

CALLING PROGRAMS:

PHASE6

DESCRIPTION:

Subroutine PHASE7, the only routine of NCAP's final phase, controls generator modification. It scans the driver file for generator modifications which are represented by a MODE=3 (generator) driver record followed by a MODE=21 (generator

modification) driver record. Unlike other device modifications, generator modification does not involve data replacement. Instead, the original generator description is disabled by writing a no-operation (MODE=2) record in its place in the driver file, and the modified generator is activated by changing its MODE from 21 to 3.

The driver file scan begins by initializing the driver file record number to DRVREC=1. A driver record is read from file 21 by subroutine DRIVRD, and subroutine PHASE7 responds to the MODE of the record as follows:

MODE=1 Driver File Header

The modify switch is turned off by MODFY=0, indicating that no modify functions have been processed. The file 20 storage address is reset by MISC(1)=MISC(4), allowing the admittance matrices and transfer functions for the next circuit analysis to be written over those from the last analysis. The updated driver file header is rewritten by subroutine DRIVWR and control returns to the driver file scan.

MODE=3 Generator

The generator driver record number is saved by JRECO=DRVREC-1 and the number of frequency sweep iterations for

AD-A076 317 SYRACUSE UNIV N Y F/G 9/5
NONLINEAR CIRCUIT ANALYSIS PROGRAM (NCAP) DOCUMENTATION. VOLUME--ETC(U)
SEP 79 J B VALENTE , S STRATAKOS F30602-79-C-0011
UNCLASSIFIED RADC-TR-79-245-VOL-3 NL

8 OF 8
ADA 076317



END
DATE FILMED
11 -79
DDC

the generator is set by NIT=MISC(6)+1. Control returns to the driver file scan.

MODE=21 Generator Modification

The modify switch is turned on by MODFY=1, indicating that a modify function has been encountered. The original generator is disabled by writing no-operation (MODE=2) records in place of the generator driver record and any sweep driver records associated with it. The no-operation records are written to file 21 by subroutine DRIVWR beginning at DRVREC=JRECO and proceeding through the next successive NIT records.

The generator modification driver record is read by subroutine DRIVRD. The modified generator is activated by setting MODE=3, and the updated driver record is rewritten by subroutine DRIVWR. In order to by-pass any sweep or modify driver records associated with the new generator, a secondary driver file scan is initiated by calling subroutine DRIVRD to read the record immediately following the new generator in the driver file. If MODE=12 (sweep) or MODE=21 (generator modification), control remains within the secondary scan. For any other value of MODE, control returns to the standard driver file scan.

MODE=22 Plot

The curve count is set to NCURV=1 in preparation for a new set of curves for the modified frequency values. MISC(5)=Ø causes new plot parameters to be calculated during the next circuit analysis. The updated plot driver record is rewritten to file 21 by subroutine DRIVWR and control returns to the driver file scan.

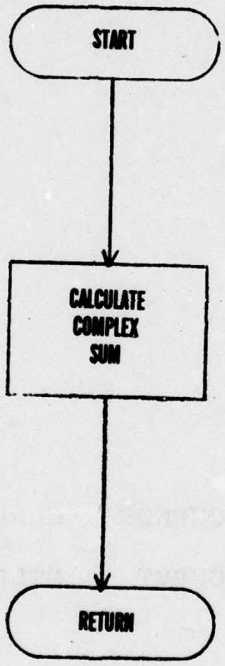
MODE=16 End

If the modify switch is on (MODFY=1), a new circuit analysis is initiated by calling subroutine PHASE1. If the modify switch is off (MODFY=Ø), program execution ends with a normal termination.

For other values of MODE, control returns to the driver file scan without further processing by subroutine PHASE7.

UTILITY

CXADD
CXDIV
CXMPY
CXPOL
CXSUB
DATARD
DATAWR
DRIVRD
DRIVWR
ERROR
FUN
LSHIFT
RSHIFT
SSCOD2
TFRD
TFWR
TIMOUT


START
CALCULATE
COMPLEX
SUM
RETURN

NAME: CXADD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates the complex sum C = A + B

VARIABLES

A = Complex addend

B = Complex addend

C = Complex sum

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

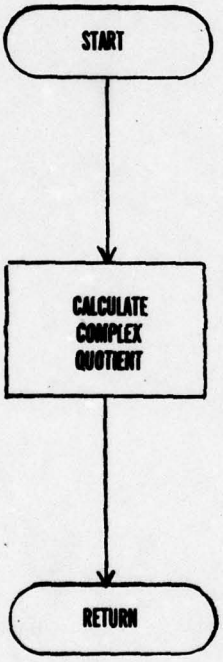
CURGEN	CURJF	CURMC	CURNDS	CURPT
CURSD	CURT	CURVD	CURVT	MTLDS
MTNDS	ZIADD			

DESCRIPTION:

Subroutine CXADD calculates the complex sum C=A+B. Each complex number is represented by two decimal numbers, the first for its real part and the second for its imaginary part. The complex sum is defined according to.

CXADD
1 of 1

```
C(1)=A(1)+B(1)
C(2)=A(2)+B(2)
```


START
CALCULATE
COMPLEX
QUOTIENT
RETURN

NAME: CXDIV

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates the complex quotient C = A/B

VARIABLES:

A = Complex dividend

B = Complex divisor

C = Complex quotient

D = Intermediate storage

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

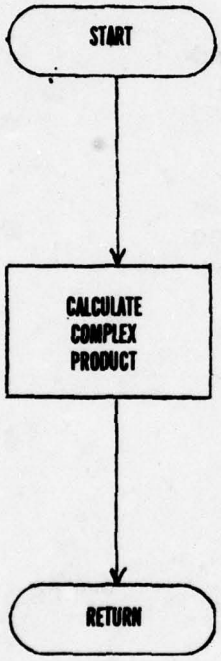
CUBIC CENIN

DESCRIPTION:

Subroutine CXDIV calculates the complex quotient C=A/B. Each complex number is represented by two decimal numbers, the first for its real part and the second for its imaginary part. The complex quotient is defined according to:

```
C(1)=(A(1)*B(1)+A(2)*B(2))/((B(1)*B(1)+B(2)*B(2))

C(2)=(A(2)*B(1)-A(1)*B(2))/((B(1)*B(1)+B(2)*B(2))
```


START
CALCULATE
COMPLEX
PRODUCT
RETURN

NAME: CXMPY

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates the complex product C = A*B

VARIABLES:

A = Complex multiplicand

B = Complex multiplier

C = Complex product

D = Intermediate storage

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

CPOSS CUFNC CURFDS CURT MTLDS

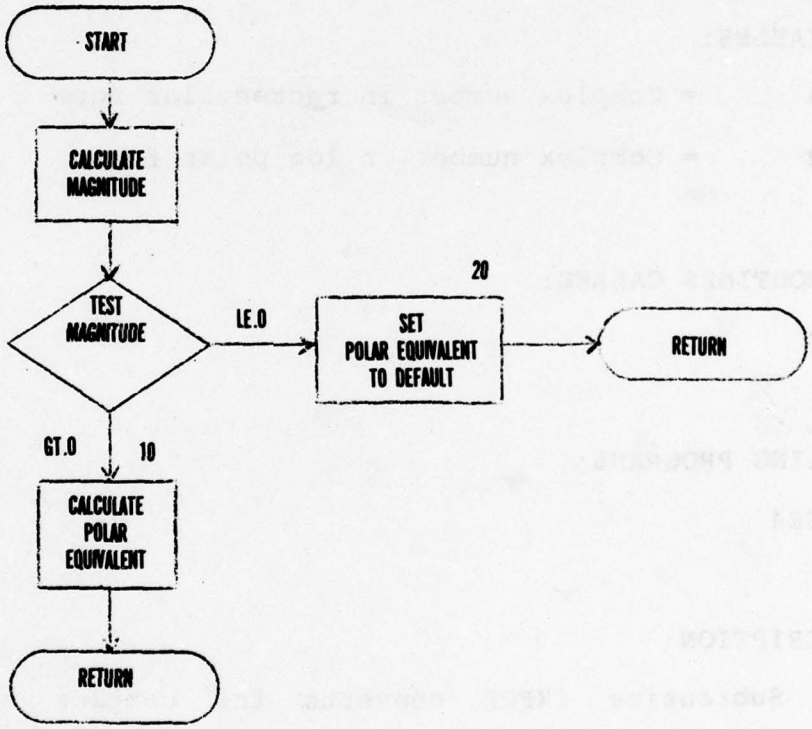
MTNDS PHASE4 TFTCNV

DESCRIPTION:

Subroutine CXMPY calculates the complex product C=A*B. Each complex number is represented by two decimal numbers, the first for its real part and the second for its imaginary part. The complex product is defined according to:

```
C(1)=A(1)*B(1)-A(2)*B(2)

C(2)=A(2)*B(1)+A(1)*B(2)
```


START
CALCULATE MAGNITUDE
TEST MAGNITUDE
LE.0
20
SET POLAR EQUIVALENT TO DEFAULT
RETURN
GT.0
10
CALCULATE POLAR EQUIVALENT
RETURN

NAME: CXPOL

TYPE: SUBROUTINE

GENERAL PURPOSE:

Converts the complex number A to log polar form B

VARIABLES:

A = Complex number in rectangular form

B = Complex number in log polar form

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

PHASE4

DESCRIPTION·

Subroutine CXPOL converts the complex number A from rectangular to log polar form. In rectangular form, the real part of the complex number is stored in A(1) and the imaginary part in A(2). The polar coordinates are returned to the calling program in B, where the radius is stored in B(1) and the angle in B(2). The conversion is carried out according to·

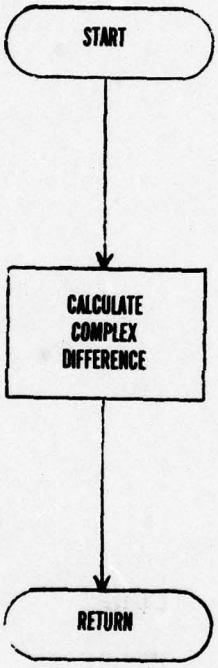
$$P(1)=20 \log_{10} \sqrt{A(1)^2 + A(2)^2}$$

$$P(2)=\text{Arctan } (A(2)/A(1))*57.2957795131$$

or if $\sqrt{A(1)^2 + A(2)^2} = 0$

P(1)=-1.0E20

P(2)=0.


START
CALCULATE
COMPLEX
DIFFERENCE
RETURN

NAME: CXSUB

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates the complex difference C = A-B

VARIABLES:

A = Complex minuend

B = Complex subtrahend

C = Complex difference

SUBROUTINES CALLED:

NONE

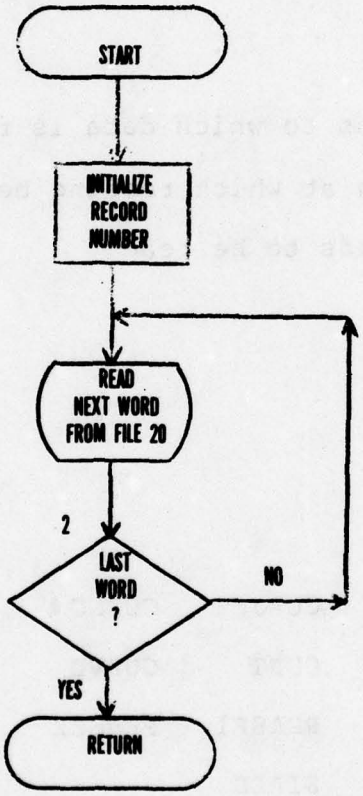
CALLING PROGRAMS:

CURCEN	CURJF	CURNC	CURNDS	CURPT
CURSD	CURT	CURVD	CURVT	MTLDS
MTNDS	ZIADD			

DESCRIPTION:

Subroutine CXSUB calculates the complex difference C=A-B. Each complex number is represented by two decimal numbers, the first for its real part and the second for its imaginary part. The complex difference is defined according to:

```
C(1)=A(1)-B(1)
C(2)=A(2)-B(2)
```


START
INITIALIZE RECORD NUMBER
READ NEXT WORD FROM FILE 20
2
LAST WORD ?
NO
YES
RETURN

NAME: DATARD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Reads data from disk file 20

VARIABLES:

AREA = Core locations to which data is read

IPTR = Disk location at which reading begins

N = Number of words to be read

SUBROUTINES CALLED:

NONE

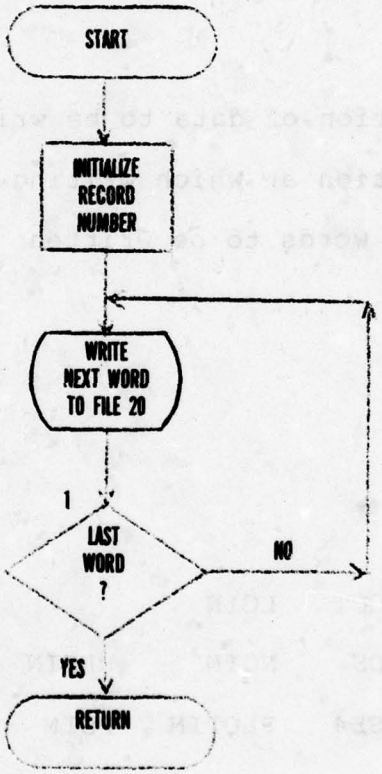
CALLING PROGRAMS:

CHNGE	CONTRL	CURGEN	CURJF	CURNC
CURNDS	CURPT	CURSD	CURT	CURVD
MODGEN	MTLDS	MTNDS	PHASE1	PHASE2
PHASE3	PHASE4	TRIANG	ZIADD	

DESCRIPTION:

Subroutine DATARD reads data from disk file 20. The transmission begins from record number IPTR and continues one word at a time until N words have been read from file 20 and stored in AREA(1)-AREA(N). The data is assumed to be decimal,

and is read without format. After the data transmission is complete, subroutine DATARD returns to the calling program.


START
INITIALIZE RECORD NUMBER
WRITE NEXT WORD TO FILE 20
1
LAST WORD ?
NO
YES
RETURN

NAME: DATAWR

TYPE: SUBROUTINE

GENERAL PURPOSE:

Writes data to disk file 20

VARIABLES:

AREA = Core location of data to be written

IPTR = Disk location at which writing begins

N = Number of words to be written

SUBROUTINES CALLED:

NONE

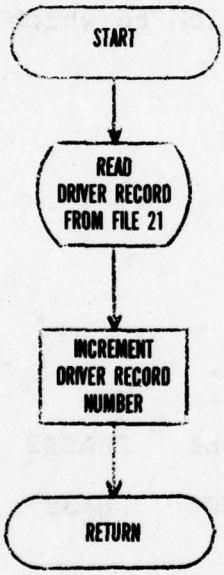
CALLING PROGRAMS:

CHNGE	DATOUT	GENOUT	LCIN	
LDSIN	MTLDS	MTNDS	NCIN	NDSIN
PHASE1	PHASE2	PHASE4	PLOTIN	PSIN
TRIANG	ZIADD			

DESCRIPTION:

Subroutine DATAWR writes data to disk file 20. The decimal values stored at AREA(1)-AREA(N) are written to file 20 one word at a time beginning at record number IPTR. The write operations are performed without format. After the data transmission is

complete, subroutine DATAWR returns to the calling program.


START
READ
DRIVER RECORD
FROM FILE 21
INCREMENT
DRIVER RECORD
NUMBER
RETURN

NAME: DRIVRD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Reads a driver record from disk file 21

VARIABLES:

LCMMN = Starting core location to which record is read

SUBROUTINES CALLED:

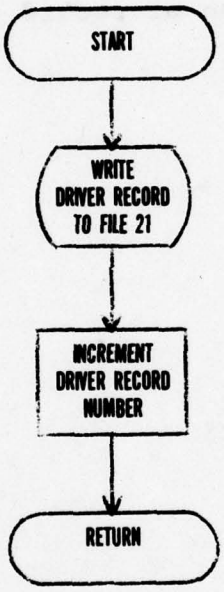
NONE

CALLING PROGRAMS:

CHNGE GENOUT MODGEN PHASE1 PHASE2
PHASE3 PHASE4 PHASE5 PHASE6 PHASE7
PRNDRV

DESCRIPTION:

Subroutine DRIVRD reads a driver record from disk file 21. The 10-word driver record stored at record number DRVREC is read from file 21 by an unformatted random access read statement and stored in the first ten words of global common. The driver record number is updated to DRVREC=DRVREC+1 and subroutine DRIVRD returns to the calling program.


START
WRITE DRIVER RECORD TO FILE 21
INCREMENT DRIVER RECORD NUMBER
RETURN

NAME: DRIVWR

TYPE: SUBROUTINE

GENERAL PURPOSE:

Writes a driver record to disk file 21

VARIABLES:

LCMMN = Starting core location of record to be written

SUBROUTINES CALLED:

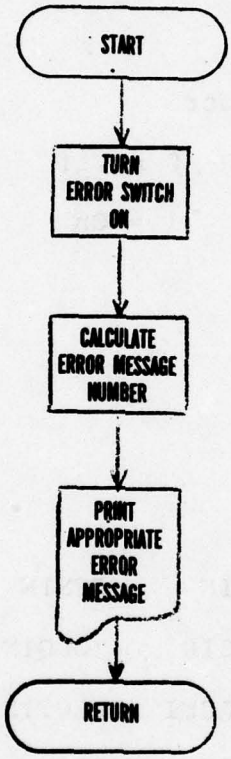
NONE

CALLING PROGRAMS:

CHNGE	DRVOUT	GENOUT	LCIN
PHASE1	PHASE2	PHASE3	PHASE5
PHASE6	PHASE7	PLOTIN	

DESCRIPTION:

Subroutine DRIVWR writes a driver record to disk file 21. The driver record stored in the first ten words of global common is written to file 21 at record number DRVREC by an unformatted random access write statement. The driver record number is updated to DRVREC=DRVREC+1 and subroutine DRIVWR returns to the calling program.


START
TURN ERROR SWITCH ON
CALCULATE ERROR MESSAGE NUMBER
PRINT APPROPRIATE ERROR MESSAGE
RETURN

NAME: ERROR

TYPE: SUBROUTINE

GENERAL PURPOSE:

Prints error messages

VARIABLES:

J = Error message number

JAPT = Input error switch: 0 = Off
1 = On

SUBROUTINES CALLED:

NONE

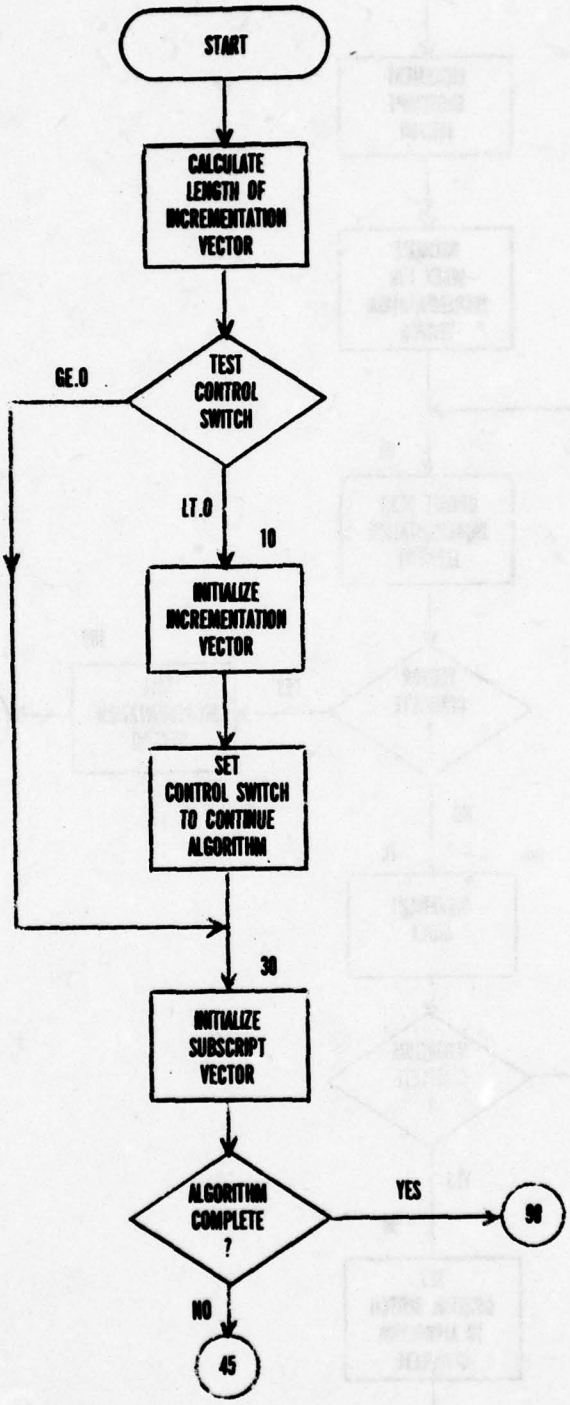
CALLING PROGRAMS:

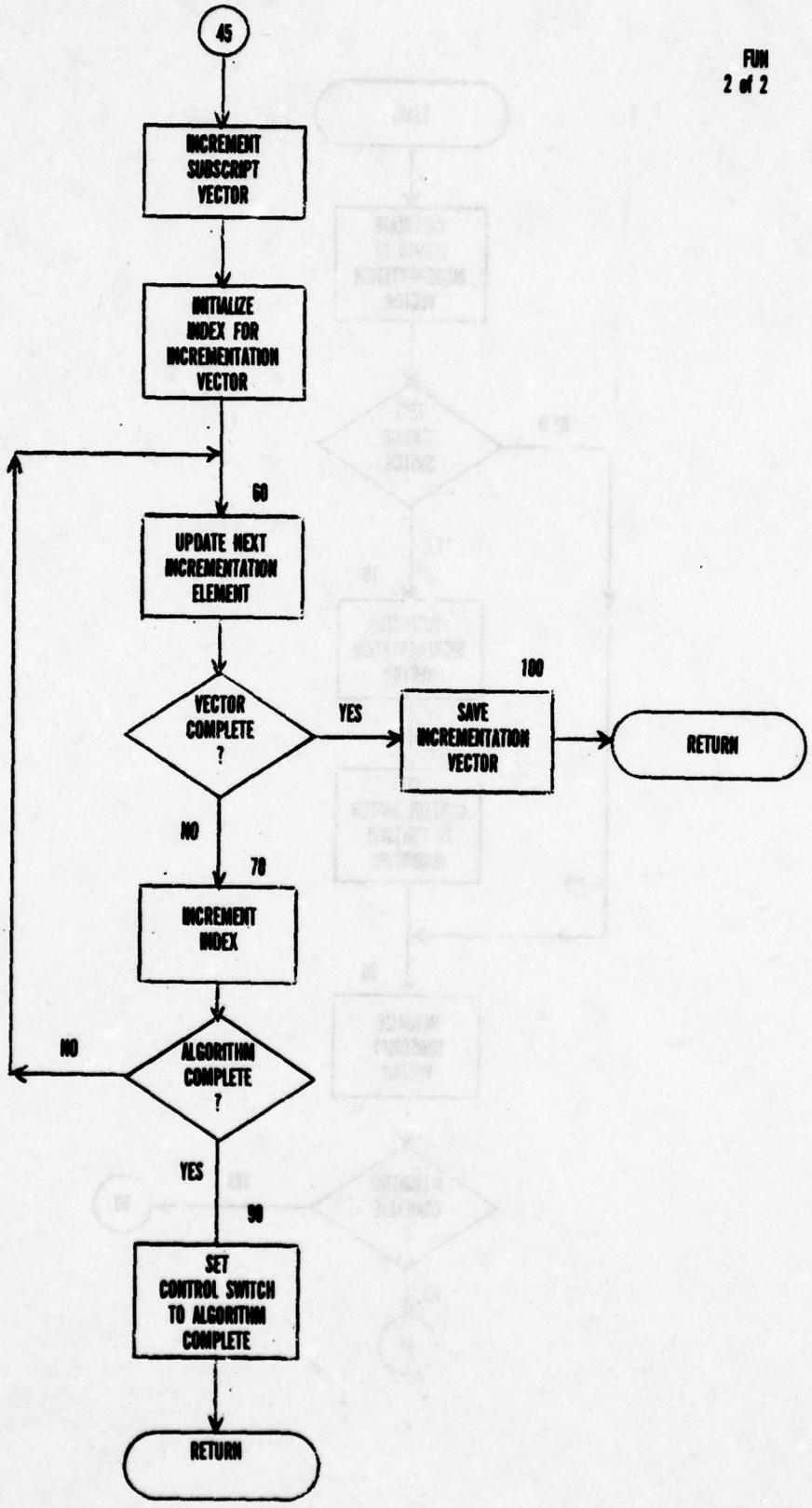
CHNGE	DATOUT	DECIF	ECIN	GENIN
GENOUT	JFETIN	LCIN	LDSIN	MODGIN
MCIN	MDSIN	MODFIN	PHASE1	PLOTIN
PROSS	PSIN	PTIN	SCIN	SDIN
TPNIN	VDIN	VTIN		

DESCRIPTION

Subroutine ERROR prints error messages. First the error switch is turned on by JAPT=1, the appropriate error message is printed according to the error number I transmitted to the

subroutine as an argument, and subroutine ERROR returns to the calling program.


START
CALCULATE LENGTH OF INCREMENTATION VECTOR
TEST CONTROL SWITCH
GE.0
LT.0
10
INITIALIZE INCREMENTATION VECTOR
SET CONTROL SWITCH TO CONTINUE ALGORITHM
30
INITIALIZE SUBSCRIPT VECTOR
ALGORITHM COMPLETE ?
YES
99
NO
45


45
INCREMENT SUBSCRIPT VECTOR
INITIALIZE INDEX FOR INCREMENTATION VECTOR
60
UPDATE NEXT INCREMENTATION ELEMENT
VECTOR COMPLETE ?
YES
100
SAVE INCREMENTATION VECTOR
RETURN
NO
70
INCREMENT INDEX
ALGORITHM COMPLETE ?
NO
YES
90
SET CONTROL SWITCH TO ALGORITHM COMPLETE
RETURN

NAME: FUN

TYPE: SUBROUTINE

GENERAL PURPOSE:

Generates subscripts of terms to be multiplied together to form one additive term of the polynomial power series expansion coefficient of order I, degree J.

VARIABLES:

I	= Order of coefficient (sum of subscripts returned)
J	= Degree of coefficient (number of subscripts returned)
V	= Subscript vector
L	= Incrementation vector
MM	= Length of incrementation vector
SWTCH	= Control switch; -1 = Initialize algorithm
	0 = Continue algorithm
	1 = Algorithm complete

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

BETA CPT CUBPT FNCTN

DESCRIPTION:

Subroutine FUN generates (indirectly) the polynomial power series expansion coefficient $F_J^I(A)$, the coefficient of X^I when the polynomial:

$$P = \sum_{n=1}^{\infty} A_n x^n$$

is raised to the Jth power. In NCAP it is convenient to denote a polynomial raised to the Jth power by:

$$P^J = \sum_{I=J}^{\infty} F_J^I(A)\, x^I$$

where $F_J^I(A)$ is a function of the coefficients A.

Although $F_J^I(A)$ cannot be formally defined mathematically, it possesses certain properties which serve as the basis for its algorithmic evaluation:

1) $F_J^I(A)$ is a sum of products of A.
2) Each product contains exactly J coefficients A.
3) The sum of the subscripts in each product is I, the exponent of X.
4) Each subscript within a product lies in the range 1 through I-J+1.

5) Each possible ordered set of subscripts of length J and sum I appears once and only once in the expansion.

6) The number of additive terms in the expansion is $\binom{I-1}{J-1}$.

Subroutine FUN does not generate the expansion coefficient directly, but rather through repeated calls produces a sequence of integer vectors K, each of which constitutes the subscripts of coefficients to be multiplied out to form a single additive term of the expansion. The calling program then forms the products and adds them to form the correct expansion.

The iterative calls to subroutine FUN are controlled by the integer variable SWTCH which is initialized to -1 by the calling program and subsequently tested and updated in FUN. Upon entry into FUN, if SWTCH is negative, the algorithm is initialized and the first subscript vector created. SWTCH is set to +1 when the last subscript vector has been formed and the algorithm is complete. For intermediate iterations SWTCH=0.

In subroutine FUN the subscripts are stored in the integer vector K. For F_J^I (A) each call to subroutine FUN creates the J subscripts which constitute one additive term of the expansion returning them to the calling program in K(1) through K(J). The subscripts are generated by applying an incrementation operation to the initial subscript vector K=(1,1,...1) of length J. This operation is controlled by an incrementation vector L(M), M=1,2,...,(I-J), where K incremented by L results in:

$$K(L(M))=K(L(M))+1 \quad \text{for } M=1,2,\ldots,(I-J)$$

The incrementation vectors are formed as follows. For the first J iterations, L(1) ranges from 1 through J and the

remaining vector elements L(2)...L(I-J)=1. For the next (J-1) iterations, L(1) ranges from 2 through J, L(2)=2, and the remaining vector elements L(3)...L(I-J)=1. This process continues until all (I-J) vector elements equal J.

The algorithm of subroutine FUN generates in K all possible ordered sets of J integers in the range 1 through (I-J+1) such that K(1)+K(2)+...+K(J)=I and can be summarized as follows:

STEP 1: Initialization

a) If (SWTCH.GE.0) go to Step 2.

b) Otherwise initialize incrementation vector:

I(N)=1 for N=1,2,...,(I-J)

c) Set SWTCH=0

STEP 2: Initialize Subscript Vector

a) Set K(N)=1 for N=1,2,...,J

b) If I=J, algorithm complete; exit with subscripts in K. Otherwise go to Step 3.

STEP 3: Increment Subscript Vector

a) Set K(L(N))=K(L(N))+1 for N=1,2,...,(I-J)

STEP 4. Form Incrementation For Next Iteration

a) Initialize index N=1

b) Update L(N)=L(N)+1. If (L(N).LE.J), incrementation vector complete go to Step 5.

c) Otherwise increment index N=N+1. If (N.LE.(I-J)), go go to Step 4b. Otherwise algorithm complete set SWTCH=1 and exit with subscripts in K.

STEP 5. Save Incrementation Vector for Next Iteration

a) Define L(M)=L(N) for M=1,N

b) Exit with subscripts in K.

A complete tabulation of $F\begin{matrix} I \\ J \end{matrix}(A)$ up to 6th order (the maximum order of analysis allowed in NCAP) is presented below.

For $F\begin{matrix} 2 \\ 2 \end{matrix}(A)$: I=2,J=2

Incrementation vector L contains (I-J)=0 elements, subscript vector K contains J=2 elements, expansion contains one additive term.

I=Inapplicable K=(1,1) ➡ A(1)*A(1)

For F_2^3 (A): I=3,J=2

Incrementation vector L contains (I-J)=1 element, subscript vector K contains J=2 elements, expansion contains two additive terms.

I=(1)	➡	K=(2,1)	➡	A(2)*A(1)
I=(2)	➡	K=(1,2)	➡	A(1)*A(2)

For F_3^3 (A): I=3,J=3

Incrementation Vector L contains (I-J)=0 elements , subscript vector K contains J=3 elements, expansion contains one additive term.

I=Inapplicable		K=(1,1,1)	➡	A(1)*A(1)*A(1)

For F_2^4 (A): I=4,J=2

Incrementation vector L contains (I-J)=2 elements, subscript vector K contains J=2 elements, expansion contains three additive terms.

L=(1,1)	➡	K=(3,1)	➡	A(3)*A(1)
I=(2,1)	➡	K=(2,2)	➡	A(2)*A(2)
I=(2,2)	➡	K=(1,3)	➡	A(1)*A(3)

For F_3^4 (A) I=4,J=3

Incrementation vector L contains (I-J)=1 element, subscript vector K contains J=3 elements, expansion contains three additive terms

I=(1) ➡ K=(2,1,1) ➡ A(2)*A(1)*A(1)

I=(2) ➡ K=(1,2,1) ➡ A(1)*A(2)*A(1)

I=(3) ➡ K=(1,1,2) ➡ A(1)*A(1)*A(2)

For F_4^4 (A): I=4,J=4

Incrementation vector L contains (I-J)=0 elements, subscript vector K contains J=4 elements, expansion contains one additive term.

L=Inapplicable K=(1,1,1,1) ➡ A(1)*A(1)*A(1)*A(1)

For F_2^5 (A): I=5,J=2

Incrementation vector L contains (I-J)=3 elements, subscript vector K contains J=2 elements, expansion contains 4 additive terms.

I=(1,1,1) ➡ K=(4,1) ➡ A(4)*A(1)

I=(2,1,1) ➡ K=(3,2) ➡ A(3)*A(2)

I=(2,2,1) ➡ K=(2,3) ➡ A(2)*A(3)

I=(2,2,2) ➡ K=(1,4) ➡ A(1)*A(4)

For F_3^5 (A) I=5,J=3

Incrementation vector L contains (I-J)=2 elements, subscript vector K contains J=3 elements, expansion contains six additive terms.

I=(1,1) ➡ K=(3,1,1) ➡ A(3)*A(1)*A(1)

L=(2,1)	➡	K=(2,2,1)	➡	A(2)*A(2)*A(1)
L=(3,1)	➡	K=(2,1,2)	➡	A(2)*A(1)*A(2)
L=(2,2)	➡	K=(1,3,1)	➡	A(1)*A(3)*A(1)
L=(3,2)	➡	K=(1,2,2)	➡	A(1)*A(2)*A(2)
I=(3,3)	➡	K=(1,1,3)	➡	A(1)*A(1)*A(3)

For F_4^5 (A): I=5,J=4

Incrementation vector L contains (I-J)=1 elements, subscript vector K contains J=4 elements, expansion contains four additive terms.

L=(1)	➡	K=(2,1,1,1)	➡	A(2)*A(1)*A(1)*A(1)
L=(2)	➡	K=(1,2,1,1)	➡	A(1)*A(2)*A(1)*A(1)
L=(3)	➡	K=(1,1,2,1)	➡	A(1)*A(1)*A(2)*A(1)
L=(4)	➡	K=(1,1,1,2)	➡	A(1)*A(1)*A(1)*A(2)

For F_5^5 (A): I=5,J=5

Incrementation vector L contains (I-J)=0 elements, subscript vector K contains J=5 elements, expansion contains one additive term.

L=INAPPLICABLE		K=(1,1,1,1,1)	➡	A(1)*A(1)*A(1)*A(1)*A(1)

For F_2^6 (A): I=6,J=2

Incrementation vector L contains (I-J)=4 elements, subscript vector K contains J=2 elements, expansion contains five addi-

tive terms.

L=(1,1,1,1) ➡	K=(5,1) ➡	A(5)*A(1)
L=(2,1,1,1) ➡	K=(4,2) ➡	A(4)*A(2)
L=(2,2,1,1) ➡	K=(3,3) ➡	A(3)*A(3)
L=(2,2,2,1) ➡	K=(2,4) ➡	A(2)*A(4)
L=(2,2,2,2) ➡	K=(1,5) ➡	A(1)*A(5)

For F_3^6 (A): I=6,J=3

Incrementation vector L contains (I-J)=3 elements, subscript vector K contains J=3 elements, expansion contains ten additive terms.

L=(1,1,1) ➡	K=(4,1,1) ➡	A(4)*A(1)*A(1)
L=(2,1,1) ➡	K=(3,2,1) ➡	A(3)*A(2)*A(1)
L=(3,1,1) ➡	K=(3,1,2) ➡	A(3)*A(1)*A(2)
L=(2,2,1) ➡	K=(2,3,1) ➡	A(2)*A(3)*A(1)
L=(3,2,1) ➡	K=(2,2,2) ➡	A(2)*A(2)*A(2)
L=(3,3,1) ➡	K=(2,1,3) ➡	A(2)*A(1)*A(3)
L=(2,2,2) ➡	K=(1,4,1) ➡	A(1)*A(4)*A(1)
L=(3,2,2) ➡	K=(1,3,2) ➡	A(1)*A(3)*A(2)
L=(3,3,2) ➡	K=(1,2,3) ➡	A(1)*A(2)*A(3)
L=(3,3,3) ➡	K=(1,1,4) ➡	A(1)*A(1)*A(4)

For F_4^6 (A): I=6,J=4

Incrementation vector L contains (I-J)=2 elements, subscript vector K contains J=4 elements, expansion contains ten additive terms.

L=(1,1)	➡	K=(3,1,1,1)	➡	A(3)*A(1)*A(1)*A(1)
L=(2,1)	➡	K=(2,2,1,1)	➡	A(2)*A(2)*A(1)*A(1)
L=(3,1)	➡	K=(2,1,2,1)	➡	A(2)*A(1)*A(2)*A(1)
L=(4,1)	➡	K=(2,1,1,2)	➡	A(2)*A(1)*A(1)*A(2)
L=(2,2)	➡	K=(1,3,1,1)	➡	A(1)*A(3)*A(1)*A(1)
L=(3,2)	➡	K=(1,2,2,1)	➡	A(1)*A(2)*A(2)*A(1)
L=(4,2)	➡	K=(1,2,1,2)	➡	A(1)*A(2)*A(1)*A(2)
L=(3,3)	➡	K=(1,1,3,1)	➡	A(1)*A(1)*A(3)*A(1)
L=(4,3)	➡	K=(1,1,2,2)	➡	A(1)*A(1)*A(2)*A(2)
L=(4,4)	➡	K=(1,1,1,3)	➡	A(1)*A(1)*A(1)*A(3)

For F_5^6 (A): I=6,J=5 Incrementation vector L contains(I-J)=1 element, subscript vector K contains J=5 elements, expansion contains five additive terms.

L=(1)	➡	K=(2,1,1,1,1,)	➡	A(2)*A(1)*A(1)*A(1)*A(1)
L=(2)	➡	K=(1,2,1,1,1,)	➡	A(1)*A(2)*A(1)*A(1)*A(1)
L=(3)	➡	K=(1,1,2,1,1,)	➡	A(1)*A(1)*A(2)*A(1)*A(1)
L=(4)	➡	K=(1,1,1,2,1,)	➡	A(1)*A(1)*A(1)*A(2)*A(1)
L=(5)	➡	K=(1,1,1,1,2,)	➡	A(1)*A(1)*A(1)*A(1)*A(2)

For F_6^6 (A): I=6,J=6 Incrementation vector L contains (I-J)=0 elements, subscript vector K contains J=6 elements, expansion contains one additive term.

L=Inapplicable K=(1,1,1,1,1,1)➡A(1)*A(1)*A(1)*A(1)*A(1)*A(1)


START
SHIFT CHARACTER RIGHT TO LEFT BY MULTIPLICATION
RETURN

NAME: LSHIFT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Shifts the character stored in the least significant bits of ICHAR to the most significant bits

VARIABLES:

NONE

SUBROUTINES CALLED:

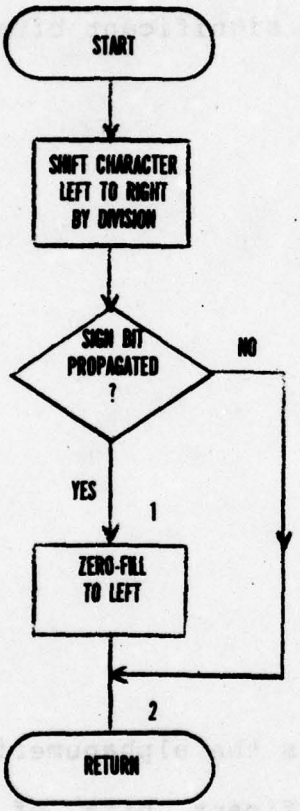
NONE

CALLING PROGRAMS:

PRSET

DESCRIPTION:

Subroutine LSHIFT shifts the alphanumeric character stored in the NBCHAR least significant bits of ICHAR to the most significant bits by multiplication by an appropriate power of 2. The resulting left-justified character is automatically zero filled to the right.


START
SHIFT CHARACTER LEFT TO RIGHT BY DIVISION
SIGN BIT PROPAGATED ?
NO
YES
1
ZERO-FILL TO LEFT
2
RETURN

NAME: RSHIFT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Shifts the character stored in the least significant bits of ICHAR to the most significant bits

VARIABLES:

NONE

SUBROUTINES CALLED:

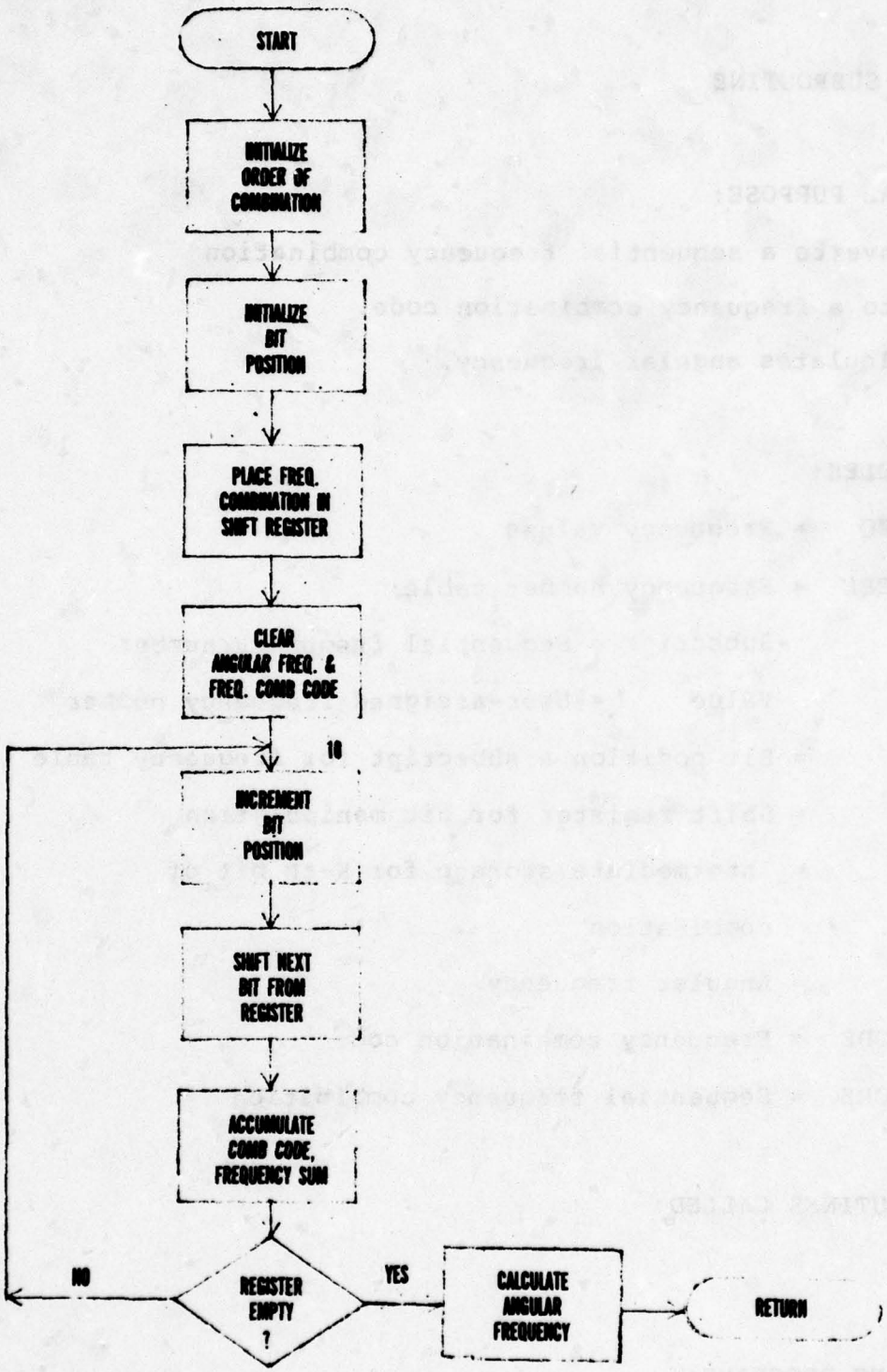
NONE

CALLING PROGRAMS:

MAIN PHASE4 SHIFT

DESCRIPTION:

Subroutine RSHIFT shifts the alphanumeric character stored in the NBCHAR most significant bits of ICHAR to the least significant bits by division by an appropriate power of 2. If the operation causes the sign bit to propagate (ICHAR.LT.0), then the word is zero filled to the left by adding $2^{NBCHAR}-1$.


START
INITIALIZE ORDER OF COMBINATION
INITIALIZE BIT POSITION
PLACE FREQ. COMBINATION IN SHIFT REGISTER
CLEAR ANGULAR FREQ. & FREQ. COMB CODE
10
INCREMENT BIT POSITION
SHIFT NEXT BIT FROM REGISTER
ACCUMULATE COMB CODE, FREQUENCY SUM
REGISTER EMPTY ?
NO
YES
CALCULATE ANGULAR FREQUENCY
RETURN

NAME: SSCOD2

TYPE: SUBROUTINE

GENERAL PURPOSE:

Converts a sequential frequency combination into a frequency combination code.
Calculates angular frequency.

VARIABLES:

FREQ = Frequency values

FRTBL = Frequency number table;
Subscript = Sequential frequency number
Value = User-assigned frequency number

K = Bit position & subscript for frequency table

L = Shift register for bit manipulation

M = Intermediate storage for K-th bit of combination

S = Angular frequency

SCODE = Frequency combination code

SCOMB = Sequential frequency combination

SUBROUTINES CALLED:

NONE

CALLING PROGRAMS:

CUPNC PHASE2

DESCRIPTION

Subroutine SSCOD2 converts a sequential frequency combination to a frequency combination code and calculates the angular frequency for that combination. Arguments transmitted to the subroutine are the secuential frequency combination, SCOMB, the frequency number table, FRTBL, and the frequency values, FRDC. Arguments returned to the calling program are SCODE and S, the calculated frequency combination code and angular frequency.

The relationship between the sequential frequency combination and the combination code is that if the Kth sequential frequency is in the combination (i.e., the Kth bit of SCOMB is 1), then the user-assigned frequency number FRTBL(K) is included in the code (i.e., the FRTBL(K)th bit of SCODE is 1). Similarly the Kth frequency value is included in the angular frequency, S, if the Kth sequential frequency appears in the frequency combination.

The method employed by subroutine SSCOD2 is to extract each secuential frequency number from SCOMB and, using the frequency number table FRTBL, to combine the corresponding user-assigned frequency numbers and values into the frequency combination code and angular frequency. The sequential frequency numbers are extracted from the combination by using a simulated shift register L, which is initially loaded with the sequential frequency combination SCOMB. A frequency number is extracted from SCOMB by shifting the right-most bit from the register into

the integer variable M. At the same time, the integer K maintains a count on the bit position being examined.

If the Kth bit is set (M=1), then the FRTBL(K)th bit of the frequency combination code is set to 1 and the Kth frequency value is added to the angular frequency. The calculations are complete when the shift register is empty.

The complete algorithm of subroutine SSCOD2 is summarized as follows:

STEP1: Initialization

a) Initialize bit position counter K=Ø and load shift register with sequential combination by L=SCOMB.

b) Clear angular frequency and combination code by S=Ø. and SCODE=Ø

STEP 2: Extract Kth Bit and Map to SCODE and S

a) Increment bit position by K=K+1

b) Shift right-most bit from register and store in M:

M=L

L=L/2

M=M-(L*2)

c) Set FTRBL(K)th bit of frequency combination code according to value of M:

SCODE=SCODE+2**(FTREL(K)-1)*M

c) Include Kth frequency in angular frequency according to value of M.

S=S*FREQ(K)*FLOAT(M)

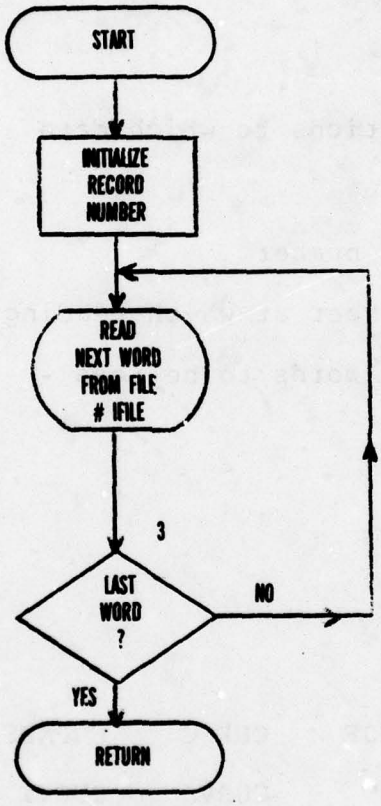
STEP 3. Test Shift Register

a) If L=0 (shift register empty), frequency combination code is complete, go to Step 4.

b) Otherwise go to Step 2.

STEP 4. Complete Angular Frequency Calculation

a) Multiply angular frequency by 2π and exit


START
INITIALIZE RECORD NUMBER
READ NEXT WORD FROM FILE # IFILE
3
LAST WORD ?
NO
YES
RETURN

NAME: TFRD

TYPE: SUBROUTINE

GENERAL PURPOSE:

Reads data from disk files 24 and 25

VARIABLES:

AREA = Core locations to which data is read

IFILE = Disk file number

IPTR = Record number at which reading begins

N = Number of words to be read

SUBROUTINES CALLED:

NONE

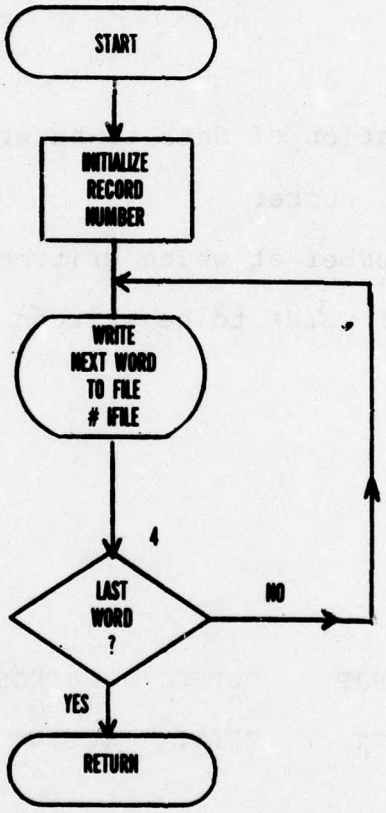
CALLING PROGRAMS:

CROSS	CURCEN	CURJF	CURNC	CURNDS
CURPT	CURSD	CURT	CURVD	CURVT
TRIANG				

DESCRIPTION:

Subroutine TFRD reads data from disk file IFILE. The transmission begins from record number IPTR and continues one word at a time until N words have been read from file IFILE and

stored in AREA(1)-AREA(N). The data is assumed to be decimal and is read without format. After the data transmission is complete subroutine TFRD returns to the calling program.


START
INITIALIZE RECORD NUMBER
WRITE NEXT WORD TO FILE # IFILE
4
LAST WORD ?
NO
YES
RETURN

NAME: TFWP

TYPE: SUBROUTINE

GENERAL PURPOSE:

Writes data to disk files 24 and 25

VARIABLES:

AREA = Core location of data to be written

IFILE = Disk file number

IPTR = Record number at which writing begins

N = Number of words to be written

SUBROUTINES CALLED:

NONE

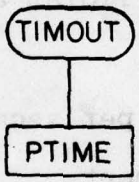
CALLING PROGRAMS:

CONTRL	CURCEN	CURJF	CURNC	CURNDS
CURPT	CURSD	CURT	CURVD	CURVT
PHASE3	TRIANG			

DESCRIPTION:

Subroutine TFWP writes data to disk file IFILE. The decimal values stored at AREA(1)-AREA(N) are written to file IFILE one word at a time beginning at record number IPTR. The write operations are performed without format. After the data

transmission is complete, subroutine TFWR returns to the calling program.


TIMOUT
PTIME

NAME: TIMOUT

TYPE: SUBROUTINE

GENERAL PURPOSE:

Calculates elapsed clock time for a single program segment and accumulates total clock time for all executions of that segment within an NCAP job

VARIABLES:

I	= Index for TIME and TTOTAL arrays; program segment identifier
TIME	= Elapsed clock time per segment
TSTART	= Start time per segment
TSTOP	= Stop time per segment
TTOTAL	= Accumulated clock time per segment

SUBROUTINES CALLED:

PTIME

CALLING PROGRAMS:

ENDIN FNCTN PHASE1 PHASE2

PHASE3 PHASE4 PHASE5 PHASE6 PHASE7

DESCRIPTION:

Subroutine TIMOUT calculates the clock time elapsed during

execution of a single program segment (a portion of a subprogram or one or more subprograms) and accumulates total clock time consumed for all executions of that segment during an NCAP job. The program segment is identified by the integer argument I, and the start time for the segment, TSTART, is transmitted to the subroutine in the labelled common area TIMEFL.

NCAP allows the timing of 30 program segments. This number may be increased or decreased by adjusting the dimension of the TSTART, TSTOP, TIME, and TTOTAL arrays in the labelled common area TIMEFL. The program segments presently selected for timing are:

Segment	Description
1	Phase 0 Complete
2	Phase 1 Complete
3	Phase 1 Generator Routines
4	Phase 1 Nonlinear Components Routines
5	Phase 1 Vacuum Diode Routines
6	Phase 1 Vacuum Triode Routines
7	Phase 1 Vacuum Pentode Routines
8	Phase 1 Transistor Routines
9	Phase 1 Semiconductor Diode Routines
10	Phase 1 JFET Routines
11	Phase 1 Linear Dependent Source Routines

12	Phase 1 Nonlinear Dependent Source Routines
13	Phase 2 Complete
14	Phase 3 Complete
15	Phase 3 Cenerator Routines
16	Phase 3 Nonlinear Components Routines
17	Phase 3 Vacuum Diode Routines
18	Phase 3 Vacuum Triode Routines
19	Phase 3 Vacuum Pentode Routines
20	Phase 3 Transistor Routines
21	Phase 3 Semiconductor Diode Routines
22	Phase 3 JFET Routines
23	Phase 3 Nonlinear Dependent Source Routines
24	Subroutine CONTRL
25	Subroutine TRIANG
26	Subroutine FNCTN
27	Phase 4 Complete
28	Phase 5 Complete
29	Phase 6 Complete
30	Phase 7 Complete

Each program segment is timed by addressing the clock before its first executable statement to acquire its TSTART time and by calling subroutine TIMOUT after its last executable statement to acquire its TSTOP time. The method of addressing the clock is a machine dependent function performed in the present NCAP version

by subroutine PTIME. Subroutine TIMOUT then calculates the elapsed time for each program segment as:

TIME(I) = TIME(I) + (TSTOP(I) - TSTART(I))

and the accumulated time as:

TTOTAL(I) = TTOTAL(I) + (TSTOP(I) - TSTART(I))

The resulting times are saved in the labelled common area TIMDEL for subsequent output by subroutine PRNTIM, and subroutine TIMOUT returns to the calling program.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.